Master of Science in Electrical Engineering

Thomas E. Flynn, BEng

Captain

Canadian Armed Forces (Air)

AFIT/GE/ENG/91S-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/91S-01

(1)

DEVELOPMENT OF A PERSONAL COMPUTER SIMULATION PROGRAM OF THE LN-94 INERTIAL NAVIGATION SYSTEM

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Thomas E. Flynn, BEng

Captain

Canadian Armed Forces (Air)

June 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1991	MS Thesis

4. TITLE AND SUBTITLE
DEVELOPMENT OF A PERSONAL COMPUTER SIMULATION PROGRAM OF THE LN-94 INERTIAL NAVIGATION SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Thomas E Flynn, Capt, CAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/91S-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis develops a simulation that emulates a Litton LN-94 INS. Real-time simulation is achieved for a stationary navigation case and a straight-and-level flight trajectory are provided to exercise the simulation dynamics. The user interface is simple and requires little previous knowledge to operate. The communication link with the MILSTD 1553B bus provide a realistic environmnet in which to collect and analyze data.

The simultion is processed on an 80386 IBM compatible personsl computer. A 23-state INS error model is implemented and integration is performed by a fifth-order Kutta-Merson routine. The simulation is interfaced to a plotting routine and data collection, both data type and rate, is controlled through the user interface. Simulation status and output is viewed from the simulation screen or on the MILSTD 1553B bus monitor.

The simulation is programmed in C programming language. Program development is discussed and flow charts of the major modules provided. The simulation validation process is outlined and results are presented.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Inertial Navigation System, Error Model, Personel Computer Simulation, Real time, MIL STD 1553B	218
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

Acknowledgements

This research effort represents a great many hours of work and a personal accomplishment I will look back upon for many years. It also represents the culmination of a graduate program unparalleled at any other academic institution. Though my name appears on the flyleaf, there are a number of others who deserve recognition in this research. Without their support and guidance, this thesis would never have been completed.

Many thanks to my thesis advisor, Capt Randy Paschall, for his patience and direction and more importantly his understanding at times when this thesis seemed like an impossible task. A special thanks to my other thesis committee members, Col Stan Lewantowicz and Dr. Peter Maybeck, for their valued contributions. Dr. Maybeck's calming approach during those moments of "panic" proved to be invaluable.

I must also thank Mr Don Smith for giving so generously of his time to pass on his knowledge of the PC and C language and to Mr Jim Hirning for his expertise in the Litton model and operation of the MILSTD 1553 communication bus.

The dedication of these people to the furtherance of higher education is exceptional. Their willingness to share their vast knowledge provides me with an asset I will rely upon for the rest of my life.

Last, but certainly not least, my deepest thanks to my wife Caroline and my very special children, Thomas, Kaila, James, and

Mallory. Their patience and constant moral support was as instrumental in the completion of this thesis as the very paper that it is written on. To my children, a promise, "my time is now your time".

Thomas E. Flynn

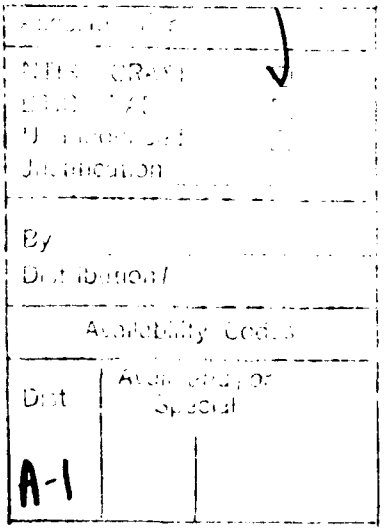


Table of Contents

List of Figures

List of Tables

Abstract

In recent years, simulation has become a very important tool in education, research and development, and training. With the advent of more complex and more expensive navigation systems, simulation has proven to be a cost-effective and reliable way to enhance the learning environment.

This thesis develops a simulation that emulates a Litton LN-94 INS. Real-time simulation is achieved for a stationary navigation case and a straight-and-level flight trajectory are provided to exercise the simulation dynamics. The user interface is simple and requires little previous knowledge to operate. The communication link with the MILSTD 1553B bus provides a realistic environment in which to collect and analyze data.

The simulation is processed on an 80386 IBM compatible personal computer. A 23-state INS error model is implemented and integration is performed by a fifth-order Kutta-Merson routine. The simulation is interfaced to a plotting routine and data collection, both data type and rate, is controlled through the user interface. Simulation status and output is viewed from the simulation screen or on the MILSTD 1553 bus monitor.

The simulation is programmed in C programming language. Program development is discussed and flow charts of the major modules provided. The simulation validation process is outlined and results are presented.

DEVELOPMENT OF A PERSONAL COMPUTER SIMULATION PROGRAM OF THE LN-94 INERTIAL NAVIGATION SYSTEM

I. Introduction

This thesis develops and implements a high fidelity LN-94 Inertial Navigation System simulation program. The simulation emulates an LN-94 23-state truth model and operates under MS DOS on an IBM compatible 386 PC. The simulation is capable of two-way communication on a MILSTD 1553B communication bus, and data input and output is controlled by a 1553B bus controller. The simulation offers the user a number of options in vehicle trajectory, state storage and plotting, and data storage. Vehicle trajectory information is provided to the simulation from an independent trajectory development program [15].

This chapter provides the reader with background information and research goals are presented in detail. The research approach is discussed and limitations in the scope of the research are indicated. An overview of the remaining chapters of the thesis is presented at the end of the chapter.

1.1 Background

The concept of simulation has proven to be a valuable aid in research, development, testing, and training. With the advances in computer technology, simulation becomes a more attractive approach since

more complex problems can be simulated in real time and with increased fidelity. In stride with these advances, avionics equipment has become much more complex, more expensive, and requires specialized equipment to support its operation. In addition, avionics equipment is difficult to obtain and very expensive to repair.

A simulation developed to work in concert with actual components can support education and research, resulting in an enhanced research and education environment. Such a simulation must be capable of operating with the same accuracy and exhibit the same operating characteristics as the original equipment. The simulation must also be capable of providing information to the user in the same format as that which is provided by the actual INS.

A situation in which a simulation would be advantageous is the Air Force Institute of Technology's Navigation Laboratory. This laboratory environment has been established to provide hands-on experience and research capability for graduate students analyzing the operation of current navigation systems. Specifically, the laboratory operates a Litton LN-94 ring laser gyro strapdown inertial navigation system (INS), a Litton LN 39 gimballed INS, a Rockwell-Collins Phase IIIA Global Positioning System (GPS) receiver, and several personal computer systems to facilitate system operations [23]. All systems are coupled to a MILSTD 1553B communication bus which is controlled by a PC-based software package and interface card developed by Digital Technology Inc [13].

The laboratory INS systems have been inoperable at various times in the past, which has led to some delays in teaching and research.

This problem highlights the need for an alternative. A simulation that can accurately emulate the operation of these systems is one solution. A high fidelity simulation of the LN-94 INS, in this case, can provide the necessary navigation information and performance as well as augment the use of actual components.

1.2 Research Objectives

The overall objective of this thesis is to develop a high fidelity PC-based simulation program emulating the performance of the LN-94 INS. The program development begins with a simulation program developed by Intermetrics Inc. of Huntington Beach CA [1]. This program is discussed in some detail in Section 2.2 of Chapter 2. Using this program as a starting point, the following sub-objectives are achieved in order to meet the overall thesis objective:

1. Verify both the level of detail of the simulation and the accuracy of the Intermetrics program.
2. Identify a truth model for the LN-94 errors, program the model, and incorporate it into the simulation.
3. Select and program a high accuracy integration routine to propagate the truth model states.
4. Program a validated random number generator to provide simulated white Gaussian noise to the stochastic truth model.
5. Develop vehicle trajectory files to allow a number of different scenarios for the simulation.

6. Ensure simulated LN-94 information provided to the 1553 data bus is in the same format as the information provided by the actual equipment.

7. Develop and implement a user interface to provide options in trajectory, data storage, and plotting.

8. Validate the simulation to determine the level of fidelity.

1.3 Research Approach

As is noted in Chapter 2, Section 2.2, the Intermetrics simulation in its original form is extremely limited as a simulation of true INS functions. It is essentially a communications shell that simulates Standard Navigation Unit (SNU) 84-1 INS message traffic based on the MILSTD 1553B protocol [4,22]. Simulated navigation performance based on spherical earth calculations with constant vehicle velocity and heading is provided. This thesis improves the capability by incorporating a 23-state LN-94 error model into the simulation. The model emulates the LN-94 operating characteristics and the simulation produces MILSTD 1553B formatted data. The simulation is validated using a number of trajectory profiles.

As clarification, the SNU 84-1 is a USAF document which issues design guidelines for an inertial navigation system based on standard form, fit, and function. The LN specification is a designator for Litton developed systems.

A 23-state LN-94 error model is used in an effort to minimize the computation overhead and at the same time achieve sufficient accuracy in the simulation. Two other alternatives were considered but deemed

unsuitable. Those included use of the full 93-state model and a 39-state model. This particular version of the LN-94 model is provided as a result of research done at AFIT by Stacey [26]. This makes the simulation dedicated to an LN-94 implementation. However, the modular nature of the program makes changing the truth model for simulating another INS a fairly simple programming task.

Numerical integration of the error model equations simulates the LN-94 error values as a function of time. Numerical accuracy is important as error values are very small, particularly early in the navigation cycle. To achieve the necessary computation accuracy, the integration routine chosen is a fifth-order routine. Double precision programming techniques are used in the integration routine.

By using the data available in Britting [8] and in the Litton Consolidated Data Requirements List (CDRL) [17], the nine basic navigation error states: three position errors, three tilt errors, and three velocity errors, can be compared for determining the simulation's accuracy. In addition, a comparison with the actual INS in static navigation is used as a measure of fidelity. The simulated results must agree in magnitude and modal character with the actual INS data, demonstrating that the error model and integration routine are valid.

Trajectory data provide the simulation with the true path of the INS. The INS truth model represents the error behavior of the INS. If the errors are summed with the trajectory information, the result is the simulated navigation solution provided by the INS.

The simulated INS information is processed to provide real-time vehicle information. Program computation time plays a critical role in

achieving real-time processing. Care is taken to optimize computation to achieve this goal.

Limiting the simulation to one trajectory would severely reduce the functionality of the simulation, so a more generalized approach is taken. To exercise the simulation, three trajectories are provided. These trajectories provide sufficient evaluation of the simulation and allow different vehicle operating environments. The flight trajectories are provided in the form of files on the host computer hard disk and are selectable from the user keyboard. The static navigation is coded into the program. The three trajectories are:

1. Static navigation
2. Straight and level with constant velocity from a predetermined initial position and heading
3. An air-to-ground fighter profile as described in the Litton CDRL [17].

A hybrid program of PROFGEN [2] known as INERCA [15] is used to generate the trajectory information. INERCA is selected based on its user interface.

The LN-94 specification that dictates the communication protocol is the Fighter Navigation Unit (FNU) 85-1 Specification [3]. To simulate the LN-94 properly, the information provided by the simulation to the 1553B communication bus must satisfy this specification. The Intermetrics simulation follows the SNU 84-1 specification. The approach here is to change the input and output specification to produce LN-94 type information in accordance with FNU 85-1.

The simulation input consists of present position latitude and longitude. The simulation output is limited to the parameters listed in Table 1.1. The simulation data is formatted as per Appendix A for use on the 1553B bus.

Table 1.1 Output Parameters

Parameter	Output (FNU 85-1 Table IV)
pitch	7
roll	8
N-S velocity	10
E-W velocity	11
vertical velocity	12
inertial altitude	13
present pos lat	14
present pos long	15
platform azimuth	104
wander angle	36
N-S acceleration	16
E-W acceleration	17
vertical acceleration	18
pitch rate (body)	33
roll rate (body)	34
yaw rate (body)	35

The user interface for any program is critically important as it normally dictates the acceptance of a program and the extent to which it is used. For this interface, a question and answer approach is used to interrogate the user. The user keys in the appropriate responses from the keyboard. All options are selected prior to executing the simulation. After the simulation starts, the user may communicate with the simulation by keyboard or 1553B bus commands.

To assist the user in getting the most information out of the simulation, the following options are programmed into the simulation:

1. type of trajectory to use
2. whether data collection is required
3. time interval between collection points
4. what data to collect (ie., which states to monitor)
5. duration of simulation

The validation approach for the overall simulation is based upon the accuracy criteria contained in FNU 85-1. By comparison of the trajectory data with the data that is provided by the simulated INS, the performance of the simulation should not exceed maximum allowable limits. Alignment is not a variable condition of this simulation, so the four minute normal gyrocompass alignment criterion shall be used. Fidelity of the simulation is measured by the ability to achieve these conditions. Exact details on the procedures for fidelity measurement are provided in Chapter 4, Section 4.2.1.

1.4 Thesis Overview

Including this introductory chapter, this thesis is comprised of five chapters and ten appendices. Chapter 2 introduces the theory that is necessary to develop the simulation program. Chapter 3 provides insight into the approaches taken in developing the simulation from a programming point of view. Chapter 4 outlines the validation process of the simulation. This includes validation of the 23-state model and comparison of the simulation performance with actual INS data. The final chapter restates the major objectives of the thesis and summarizes

to what degree the goals have been met. Validation results are summarized and recommendations and comments for future development are provided.

The appendices provide supporting figures and tables, a description of program functions, validation results, and a user's guide for the simulation. Appendices are listed as follows:

Appendix A: FNU 85-1 Word Formats

Appendix B: LN-94 Truth Model Definition

Appendix C: LN-94 Dynamics and Noise Matrices

Appendix D: Programming Flow Charts

Appendix E: Program Functions

Appendix F: Simulation Operator's Manual

Appendix G: Reduced Order Model Validation Results

Appendix H: Simulation Validation Results for Static Navigation

Appendix I: Simulation Validation Results for Straight and Level Trajectory

Appendix J: Error Behavior of Simulated Straight and Level Trajectory

II. Theory

As discussed in the Chapter 1, the primary objective of this thesis is to develop a high fidelity simulation of an LN-94 INS. The objectives have been presented and the research approach has been outlined. This chapter reviews the theory that is required to develop the simulation.

Of particular importance to this research is a discussion of the theory necessary to understand the operation of the LN-94, its error model, and the propagation techniques used. An outline of MILSTD 1553B principles and a discussion of applicable hardware are included. The chapter concludes with a summary of the software necessary to conduct this research and to execute the simulation.

2.1 LN-94 Inertial Navigation System

The INS for this research is the Litton Systems LN-94. As a matter of procedure, the term LN-94 will be used through out this theory chapter and the remainder of the document. This is noted as confusion may arise from the fact that the Litton CDRL refers to an LN-93. Operationally the two configurations are identical. The different designation lies in the fact that the LN-94 is an LN-93 that is physically reconfigured for use in the F-15.

The LN-94 is a strapdown inertial system using three single-degree-of-freedom accelerometers and three ring laser gyroscopic sensors. Strapdown refers to the concept that there is no moving gimbal

structure. There is no angular motion of components with respect to the vehicle, and the body frame represents the mechanization frame.

As is implied by the last statement, the INS implements its operation in terms of reference frames. The LN-94 frames are the accelerometer a-frame, gyro g-frame, the sensor s-frame, the body b-frame, the platform p-frame, the computer c-frame, and the true t-frame. For this implementation, the body frame, the sensor frame, and the platform frame are the same. The computer frame is the frame in which the INS computations are done and the true frame represents the actual navigation frame.

To highlight the relationship of these frames to each other a look at the transformation equations is appropriate. All INS data is computed in the computer frame and represents the INS-indicated values. To transform gyro and accelerometer data into the true frame the following transformations are used:

$$C_g^t = C_b^t \, C_s^b \, C_g^s \tag{1}$$

and

$$C_a^t = C_b^t \, C_s^b \, C_a^s \tag{2}$$

where C is the direction cosine matrix transforming vectors from the lower subscript frame to the upper subscript frame.

Since the sensitive axes of both gyros and accelerometers is nominally aligned with the body axes, except for uncalibrated installation errors, then it must be assumed that the body sensor transformation is equal to identity. All transformations from the sensor frame to the true frame can then be achieved using the body-to-

true-frame transformation. Refer to Section 2.1.1 for a description of the different coordinate frames.

The system is not without errors. These errors are represented by misalignments between frames. The true frame is offset from the computer frame by an error vector, $\delta\theta$, which represents the computation errors and the uncertainties of the INS. These errors propagate themselves in the form of latitude, longitude, and wander angle errors. The true frame is also offset from the platform frame. This offset represents the physical misalignment in the gyros and accelerometers when they are mounted to the INS.

The accelerometers provide specific force measurements experienced by the INS and the gyros provide attitude rate measurements due to body and earth rotations. Accelerometer and gyro triads provide these measurements in each of the three sensitive axes. This information is used to determine position, velocity, and attitude of the INS. A digital computer monitors the vehicle attitude based on gyro information. From this information the computer can then provide the transformation necessary to transform the specific force measurements into the computation frame [8]. As mentioned previously, the computer then computes the transformation into the true frame and useful navigation information is provided.

The LN-94 is defined as a local level wander azimuth mechanization [16]. This means that the vertical axis is uncommanded [8]. Only level axes are commanded and the wander angle represents the angle about the vertical axis between north and the wander azimuth y-axis. By monitoring this angle, the INS will always know the platform attitude

but the reference frames will be skewed by the wander angle. Since the LN-94 is a strapdown INS, commanding the level axes is a software function. This implementation is particularly useful in the polar regions as the vertical axis is insensitive to azimuth gyro torquing uncertainties.

2.1.1 Reference Frames. The navigation solution is normally expressed in terms of latitude, longitude, and altitude; better known as geographical coordinates or simply map coordinates. In any navigation system it is often more useful to relate this information to the earth-centered earth-fixed reference frame (ECEF). To use this frame accurately, knowledge of the earth's physical structure and shape must be known precisely. A number of earth models have been utilized but the most accurate to date is based upon the World Geodetic System 1984 (WGS-84). The system models the earth as an oblate spheroid defined by the terms in Table 2.1 [12]. The system defines the ECEF frame as follows:

x_e- lies in the equatorial plane and intersects the Greenwich Meridian

y_e- lies in the equatorial plane and rotated east, orthogonal to the z_e axis

z_e- coincident to the earths rotational axis and pointing north in the standard right-hand orthogonal

The ECEF frame is only one of the frames used to develop an INS navigation solution. The INS utilizes other frames such as the wander azimuth frame (in the case of the LN-94) and the body frame. To understand the wander azimuth concept and other pertinent frames, a summary of the system reference frames follows:

Table 2.1 WGS-84 Parameters

Parameter	Definition	Value
ω_{ie}	Angular rate of Earth	7.292115e-5 s^{-1}
a	Semi-major Axis (Equatorial Radius)	6378137 m
b	Semi-minor Axis (Polar Radius)	6356752.3142 m
e	First Eccentricity	0.0818191908426
f	Flattening (Ellipticity)	0.00335281066474
g_o	Equatorial Acceleration of Gravity	9.7803267714 ms^{-2} 32.087686258 fs^{-2}

1. Earth Centered Earth Fixed (ECEF)- The ECEF e-frame of the LN-94 is different from the standard WGS-84 convention. The Litton ECEF frame shown in Figure 2.1, has its origin at the center of the earth with the z_e axis at the intersection of the equatorial plane on the Greenwich meridian and the y_e axis aligned with the north pole. The x_e axis is orthogonal to this plane and intersecting the equator.

2. Navigation Frame- The navigation n-frame is oriented in an East, North, Up (ENU) convention with the U axis pointing in the direction of the local vertical as shown in Figure 2.1.

3. True Frame- This t-frame represents the actual latitude and longitude of the INS. The true frame is rotated about the U axis in a counterclockwise direction at an angle, α, referred to as the wander angle. When this angle is zero the navigation frame and the true frame are coincident in the $[x,y,z]_t = [E,N,U]_t$

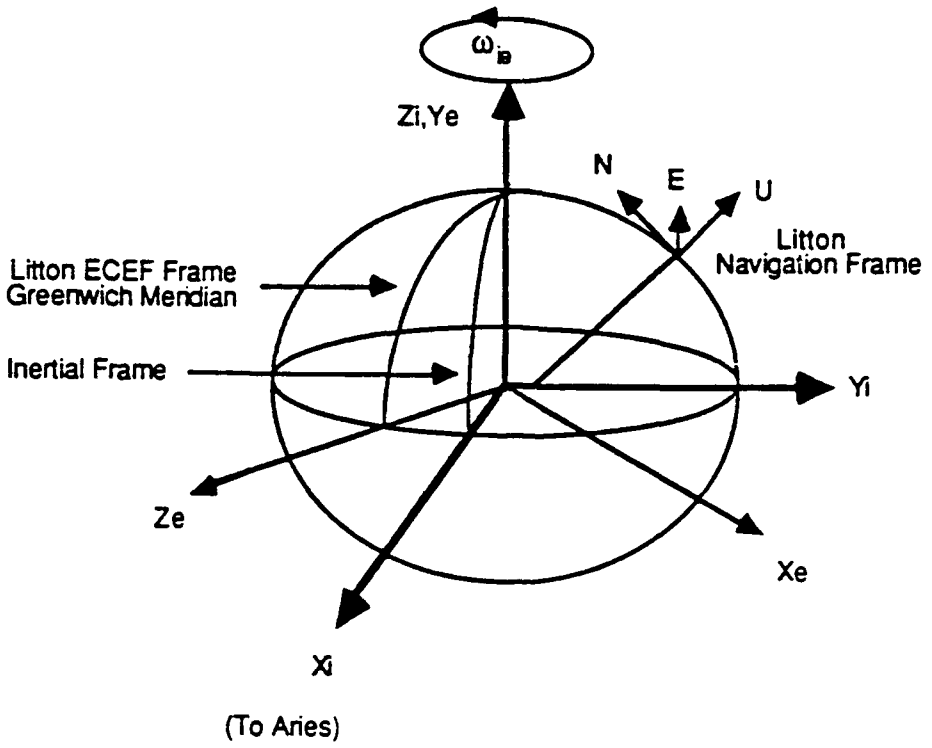


Figure 2.1 Litton ECEF and Navigation Reference Frames

convention. For this application the wander angle will generally not be zero. Refer back to Section 2.1 for details on wander angle mechanization. In the FNU 85-1 specification, reference is made to the wander azimuth frame while in the Litton documentation, reference is made to the true frame. These frames are the same.

4. Platform Frame- The platform p-frame is coincident with the true frame in an errorless system. Typically, however, the frames are misaligned by small attitude angles. This three dimensional misalignment is represented by ϕ_x, ϕ_y, and ϕ_z as shown in Figure 2.2 [17]. Refer to Section 2.1.2 for the matrix definition

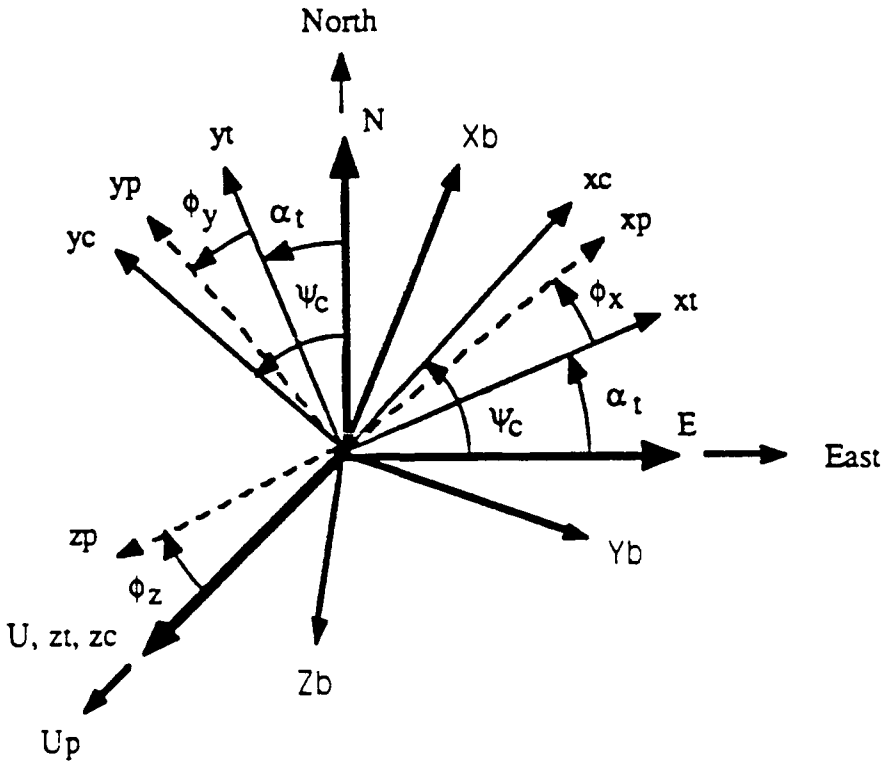


Figure 2.2 Navigation Reference Frames (t-true, p-platform, c-computer, b-body) $\phi_{x,y,z}$ Tilt Errors, α_t and ψ_c Wander Angles

of these angles. The misalignments indicated above are actually representative of the physical misalignments of the components as experienced by the true frame when transformed from the gyro and accelerometer frames.

5. Computer Frame- The computer c-frame represent latitude and longitude as indicated by the INS. As in the other frames, when the wander angle is zero, this frame is coincident with the navigation frame as shown in Figure 2.2. As discussed previously, the values computed by this frame differ from the quantities in

the true frame because of computational and physical uncertainties.

6. Body Frame- The body b-frame, Figure 2.2, is a function of the orientation of the vehicle containing the INS. For an aircraft application, the x_b axis is parallel to the fuselage in the direction of vehicle nose. The y_b axis is parallel to the right wing and positive in that direction. The z_b axis is positive out the bottom of the fuselage to form an orthogonal right-hand convention frame [3].

2.1.2 Coordinate Transformations. One of the key principles of inertial navigation is the transformation of vectors from one reference frame to another. One means of doing this is using the direction cosine matrix (DCM). Section 2.1 discusses a number of these transformations and the following is a definition of these and some of the other applicable transformations. Note that the true frame convention rather than wander azimuth frame convention is being used to make this overview of the error model more compatible with the discussion in the Litton document [17].

For the ECEF and the true frames [3]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^e = C_t^e \begin{bmatrix} x \\ y \\ z \end{bmatrix}^t \tag{3}$$

$$C_t^e = \begin{bmatrix} \cos\lambda\cos\alpha - \sin L\sin\lambda\sin\alpha & -(\cos\lambda\sin\alpha + \sin L\sin\lambda\cos\alpha) & \sin\lambda\cos L \\ \cos L\sin\alpha & \cos L\cos\alpha & \sin L \\ (\sin\lambda\cos\alpha + \cos\lambda\sin L\sin\alpha) & \sin\lambda\sin\alpha - \cos\lambda\sin L\cos\alpha & \cos\lambda\cos L \end{bmatrix} \tag{4}$$

where:

λ = terrestrial longitude
L = geodetic latitude
α = wander angle

For the ECEF and navigation frame [3]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^{e} = C_{n}^{e} \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{n} \tag{5}$$

$$C_{n}^{e} = \begin{bmatrix} \cos\lambda & -\sin\lambda\sin L & \sin\lambda\cos L \\ 0 & \cos L & \sin L \\ -\sin\lambda & -\cos\lambda\sin L & \cos\lambda\cos L \end{bmatrix} \tag{6}$$

where:

λ = terrestrial longitude
L = geodetic latitude

For navigation and true frames:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^{n} = C_{t}^{n} \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{t} \tag{7}$$

where:

$$C_{n}^{t} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Note that the only misalignment between the navigation and true frames is the wander angle, α.

For the true and body frames [3]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^{b} = C_{t}^{b} \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{t} \tag{9}$$

$$C_t^b = \begin{bmatrix} \cos\theta\sin\psi & \cos\theta\cos\psi & \sin\theta \\ \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & -\sin\phi\cos\theta \\ \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi & \cos\phi\sin\theta\cos\psi+\sin\phi\cos\psi & -\cos\phi\cos\theta \end{bmatrix} \quad (10)$$

where:

ϕ = roll
θ = pitch
ψ = platform azimuth

Since the sensitive axes of both the gyros and the accelerometers are considered to represent the body axis of the vehicle for a strapdown mechanization, the transformation from the sensor frame to the navigation frame is represented by the body-to-true-frame transformation matrix [3].

In the case of the platform and the true frames, recall that the only difference is a small attitude error where ϕ is a skew-symmetric matrix representing the small misalignment. In this case the transformation is [17]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^p = [I+\phi] \begin{bmatrix} x \\ y \\ z \end{bmatrix}^t \quad (11)$$

where:

$$\phi = \begin{bmatrix} 0 & \phi_z & -\phi_y \\ -\phi_z & 0 & \phi_x \\ \phi_y & -\phi_x & 0 \end{bmatrix} \quad (12)$$

2.1.3 LN-94 Error Model. This section presents a brief discussion of the LN-94 error model and identifies the system states used in this thesis. The reader is referred to the Litton CDRL [17] and

to Hirning [14] and Stacey [26] for a more rigorous analysis of the model.

The INS error states change much more slowly than the system navigation states for an INS in a dynamic environment. Consequently, a longer sampling rate can be used and the computational load is minimized because these error states can be linearized. In addition, slower changing states allow for simplification of some of the propagation equations and again assist in reducing computational loading.

The linearized INS error state equation is of the form:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{F}(t)\delta\mathbf{x}(t) + \mathbf{G}(t)\mathbf{w}(t) \tag{13}$$

where:

$$\mathbf{F}(t) = \left.\frac{\partial \mathbf{F}[x, u(t), t]}{\partial x}\right|_{x=x_n} \tag{14}$$

$\delta\mathbf{x}(t)$ is a time-varying state error vector, $\mathbf{F}(t)$ is a time-varying system dynamics matrix, $\mathbf{w}$ is a white noise vector with $\mathbf{G}=\mathbf{I}$, and x_n is the nominal value of x in the linearization process.

The LN-94 error model presented in Litton's CDRL has a 93-state error vector. $\delta\mathbf{x}$ is partitioned into six smaller error vectors for convenience of analysis. The vectors are defined as follows:

$$\delta\mathbf{x} = \left[\, \delta\mathbf{x}_1^T \;\; \delta\mathbf{x}_2^T \;\; \delta\mathbf{x}_3^T \;\; \delta\mathbf{x}_4^T \;\; \delta\mathbf{x}_5^T \;\; \delta\mathbf{x}_6^T \,\right]^T \tag{15}$$

where:

$\delta\mathbf{x}_1$ represents position, velocity, attitude, and vertical channel errors (13 states).
$\delta\mathbf{x}_2$ represents gyro, accelerometer, and barometer correlated errors (16 states).

δx_3 represents gyro bias errors (18 states).
δx_4 represents accelerometer bias and barometer bias errors (22 states).
δx_5 represents accelerometer and gyro initial thermal transients (6 states).
δx_6 represents the gyro compliance errors (18 states).

Definition of each of the 93 states is found in Appendix B. In state space matrix form, Equation (13) is explicitly represented by the following equation [17]:

$$\begin{bmatrix} \delta\dot{x}_1 \\ \delta\dot{x}_2 \\ \delta\dot{x}_3 \\ \delta\dot{x}_4 \\ \delta\dot{x}_5 \\ \delta\dot{x}_6 \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{14} & F_{15} & F_{16} \\ 0 & F_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & F_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \delta x_3 \\ \delta x_4 \\ \delta x_5 \\ \delta x_6 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{16}$$

The dynamics matrix, F, consisting of eight non-zero sub-matrices, represents the homogeneous dynamics of the system. In this case, the dynamic matrix is a linearized, time-varying, 93x93 matrix. The reader is referred to Appendix C for the definition of the full dynamics matrix.

The driving noise vector, Gw, is a Gaussian white noise function of the form [17]:

$$Gw = \begin{bmatrix} w_1^T & w_2^T & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{17}$$

The autocorrelation kernel functions associated with the two white noise matrices are [18]:

$$E(w_1(t)w_1^T(t+\tau)) = Q_{11}(t)\delta(t-\tau) \tag{18}$$

$$E\{\mathbf{w}_2(t)\mathbf{w}_2^T(t+\tau)\} = Q_{22}(t)\delta(t-\tau) \tag{19}$$

where $\delta(t-\tau)$ is the dirac delta function and Q_{11} and Q_{22} represent the white noise strengths. The form of the Q matrix is [17]:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

The definition of the 93-state model Q matrix is found in Appendix C.

Computationally, this is a very large truth model. Ideally, if a model could be developed with fewer error states and essentially the same performance, the job of propagating this model would be much less tasking. Research to establish an effective reduced order truth model has indicated that a model on the order of 23 states is acceptable for the same analysis and applications [23,27]. The most significant partition is $\delta\mathbf{x}_1$ which contains the basic nine error states defined in Britting [8] and the necessary baro-aiding error states. The addition of the first ten states of $\delta\mathbf{x}_2$ accounts for gyro drift, accelerometer induced errors, gravity vector errors, and the barometric noise error. This model, adjusted with appropriate noise strengths, performs well compared to the Litton documentation [17,27].

This 23-state error model is of the same form as the 93-state error model. The position error differential equations are:

$$\delta\dot{\theta}_x = -\rho_y\delta\theta_z - C_{RY}\delta V_y \tag{21}$$

$$\delta\dot{\theta}_y = \rho_x \delta\theta_z + C_{RX}\delta V_x \tag{22}$$

$$\delta\dot{\theta}_z = \rho_y \delta\theta_x - \rho_x \delta\theta_y \tag{23}$$

$\delta\theta_{x,y,z}$ are components of $\delta\theta$ which define the misalignment from the true frame to the computer frame. The craft rate, ρ, defined as the angular rate of the navigation reference frame with respect to earth, has the following components:

$$\rho_x = -V_y C_{RY} \tag{24}$$

$$\rho_y = V_x C_{RX} \tag{25}$$

$$\rho_z = 0 \tag{26}$$

The components of earth spheroid inverse radii of curvature are:

$$C_{RX} = \frac{1}{a}\left[1 - \frac{h}{a} - f\left(C_e^t(2,3)^2 - 2C_e^t(2,1)^2\right)\right] \tag{27}$$

$$C_{RY} = \frac{1}{a}\left[1 - \frac{h}{a} - f\left(C_e^t(2,3)^2 - 2C_e^t(2,2)^2\right)\right] \tag{28}$$

where the equatorial radius, a, and the earth's ellipticity, f, are defined in Table 2.1. Vehicle altitude corresponds to the term h.

$V_{x,y,z}$ are components of the vehicle velocity with respect to earth coordinates, and $\delta V_{x,y,z}$ are errors in the computed velocity with respect to the true velocity.

The platform tilt differential equations are:

$$\dot{\phi}_x = -\Omega_z \delta\theta_y + \Omega_y \delta\theta_z + \omega_z \phi_y - \omega_y \phi_z - C_{RY}\delta V_y + b_{x_c} + \eta_{b_x} \tag{29}$$

$$\dot{\phi}_y = \Omega_z \delta\theta_x - \Omega_x \delta\theta_z - \omega_z \phi_x + \omega_x \phi_z + C_{RX} \delta V_x + b_{y_c} + \eta_{b_y} \tag{30}$$

$$\dot{\phi}_z = -\Omega_y \delta\theta_x + \Omega_x \delta\theta_y + \omega_y \phi_x - \omega_x \phi_y + b_{z_c} + \eta_{b_z} \tag{31}$$

where the earth sidereal rate, Ω, more often referred to as ω_{ie}, is computed as follows:

$$\begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} = \begin{bmatrix} \omega_{iex} \\ \omega_{iey} \\ \omega_{iez} \end{bmatrix} = \begin{bmatrix} \omega_{ie} C_e^t(2,1) \\ \omega_{ie} C_e^t(2,2) \\ \omega_{ie} C_e^t(2,3) \end{bmatrix} \tag{32}$$

and the angular rate of the navigation frame with respect to inertial space, referred to as spatial rate is defined as follows:

$$\omega_x = \rho_x + \Omega_x \tag{33}$$

$$\omega_y = \rho_y + \Omega_y \tag{34}$$

$$\omega_z = \rho_z + \Omega_z \tag{35}$$

The variables $b_{(x,y,z)c}$ are components of the gyro correlated drift and η is a white noise component of the applicable subscript, in this case, gyro white noise.

The velocity differential equations are discussed in terms of level velocity for two dimensional problems, or level and vertical velocity for three dimensional problems. The level velocity differential equations are:

$$\begin{aligned}\delta\dot{V}_x = &-2(V_y\Omega_y + V_z\Omega_z)\delta\theta_x + 2V_y\Omega_x\delta\theta_y + 2V_z\Omega_x\delta\theta_z \\ &- A_z\phi_y + A_y\phi_z - V_zC_{RX}\delta V_x + 2\Omega_z\delta V_y \\ &- (\rho_y + 2\Omega_y)\delta V_z + \delta g_x + \eta_{A_x}\end{aligned} \tag{36}$$

$$\begin{aligned}\delta\dot{V}_y = &\ 2V_x\Omega_y\delta\theta_x - 2(V_x\Omega_x + V_z\Omega_z)\delta\theta_y + 2V_z\Omega_y\delta\theta_z \\ &+ A_z\phi_x - A_x\phi_z - 2\Omega_z\delta V_x - V_zC_{RY}\delta V_y \\ &+ (\rho_x + 2\Omega_x)\delta V_z + \delta g_y + \eta_{A_y}\end{aligned} \tag{37}$$

The variables $\delta g_{x,y,z}$ are component errors in the gravity vector and $A_{x,y,z}$ are components of specific force.

Vertical velocity is a more complex problem, given that the model must account for vehicle acceleration as well as gravitational acceleration. The introduction of vertical channel gains, k_1, k_2, k_3, and k_4 to the model (Equations (47) to (50)) plus the more complex effects of the earth's physical shape, contribute to this complexity. Vertical channel differential equations are defined as follows:

$$\begin{aligned}\delta\dot{V}_z = &\ 2V_x\Omega_z\delta\theta_x + 2V_y\Omega_z\delta\theta_y - 2(V_y\Omega_y + V_x\Omega_x)\delta\theta_z \\ &- A_y\phi_x + A_x\phi_y + (\rho_y + 2\Omega_y + V_xC_{RX})\delta V_x \\ &- (\rho_x + 2\Omega_x - V_yC_{RY})\delta V_y + (2g_o/a)\delta h \\ &+ k_2\delta h_c - k_2\delta h_L + k_2\delta S_4 - \delta S_3 + \delta g_z + \eta_{A_z}\end{aligned} \tag{38}$$

$$\delta\dot{h} = \delta V_z - k_1\delta h_L + (k_1 - 1)\delta S_4 + k_1\delta h_c \tag{39}$$

$$\delta\dot{h}_L = \delta h - \delta h_L \tag{40}$$

$$\delta\dot{S}_3 = k_3\delta h_L - k_3\delta S_4 - k_3\delta h_c \tag{41}$$

$$\delta\dot{S}_4 = (k_4 - 1)\delta S_4 + k_4\delta\dot{h}_L - k_4\delta\dot{h}_c \tag{42}$$

$$\delta\dot{h}_c = (-\beta\delta h_c)\ \delta h_c \tag{43}$$

$$\delta\dot{h}_c = \delta\dot{h}_B\ ,\delta h_c = \delta h_B \tag{44}$$

where:

δh is error in vehicle altitude
δh_L is error in lagged inertial altitude
δh_c is barometer correlated bias noise error
δS_3 is error in vertical aiding
δS_4 is error in vertical aiding
δh_B is barometric altitude error
$\beta\delta h_c$ is baro inverse correlation time

The reader is referred to Figure 2.3 for further clarification of the vertical channel equations.

Specific force measurements are not a part of the model. Specific force information is obtained from the following transformation:

$$\begin{bmatrix} A_x^t \\ A_y^t \\ A_z^t \end{bmatrix} = C_b^t \begin{bmatrix} A_x^b \\ A_y^b \\ A_z^b \end{bmatrix} \tag{45}$$

Measurements for the simulation are provided by trajectory data.

The vertical channel gains are calculated based on the following equations:

$$\lambda_1 = 100 \left[1 + \left[\frac{\Delta}{\Delta_o}\right]^2\right] \tag{46}$$

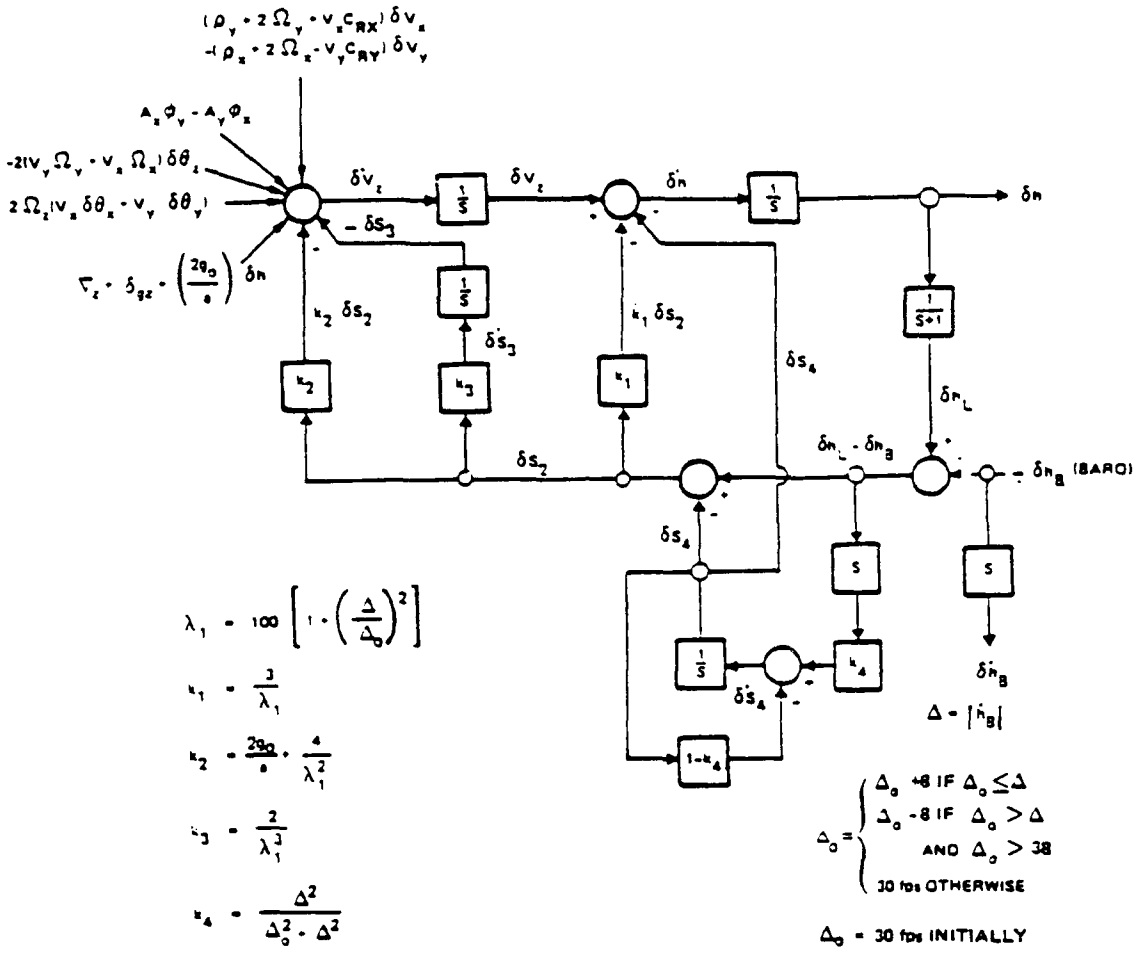


Figure 2.3 Block Diagram of Vertical Channel Error Model [17]

$$k_1 = \frac{3}{\lambda_1} \tag{47}$$

$$k_2 = \frac{2g_o}{a} + \frac{4}{\lambda_1^2} \tag{48}$$

$$k_3 = \frac{2}{\lambda_1^3} \tag{49}$$

$$k_4 = \frac{\Delta^2}{\Delta_o^2 + \Delta^2} \tag{50}$$

where:

$$\Delta = |\dot{h}_b| = |V_z^T| \tag{51}$$

$$\Delta_o = \begin{cases} 30 \ fps \ initially \\ \Delta_o + 8 \ \ if \ \ \Delta_o \leq \Delta \\ \Delta_o - 8 \ \ if \ \ \Delta_o > \Delta \ \ and \ \ \Delta_o > 38 \\ 30 \ fps \ otherwise \end{cases} \tag{52}$$

A summary of the error states as represented in their respective differential equations is defined in Table 2.2.

*2.1.4 **Error Model Propagation**.* The model is defined by the previous section and it now remains to discuss the method used to propagate the model and produce the information necessary for the simulation.

The goal of the simulation is to produce INS-indicated information to the simulation user. As mentioned previously, an INS is not error-free. True trajectory is defined as:

$$true\ trajectory = INS\ indicated - INS\ error \tag{53}$$

and the INS indicated value is represented as:

Table 2.2 Reduced Truth Model States

State Number	Symbol	Error Definition
1,2,3	$\delta\theta_x, \delta\theta_y, \delta\theta_z$	X,Y,Z Position
4,5,6	ϕ_x, ϕ_y, ϕ_z	X,Y,Z Platform Tilt
7,8,9	$\delta V_x, \delta V_y, \delta V_z$	X,Y,Z Velocity
10	δh	Altitude
11	δh_L	Lagged Inertial Altitude
12,13	$\delta S_3, \delta S_4$	Vertical Aiding
14,15,16	b_{xc}, b_{yc}, b_{zc}	Gyro Correlated Drift
17,18,19	$\nabla_{xc}, \nabla_{yc}, \nabla_{zc}$	Accelerometer and Velocity Correlation Noise
20,21,22	$\delta g_x, \delta g_y, \delta g_z$	Gravity
23	δh_c	Baro Correlated Bias Noise

$$\textit{INS indicated} = \textit{true trajectory} + \textit{INS error} \tag{54}$$

The object of the error model propagation is to integrate the model forward in time, and when coupled with the trajectory information at the end of the applicable interval time, to simulate the INS-indicated values as defined in Equations (53) and (54). Specifically, the INS-indicated position is defined as:

$$\begin{bmatrix} L_{ins} \\ \lambda_{ins} \\ h_{ins} \end{bmatrix} = \begin{bmatrix} L_{true}+\delta L \\ \lambda_{true}+\delta\lambda \\ h_{true}+\delta h \end{bmatrix} \tag{55}$$

where:

$$\delta L = \delta\theta_y \sin\alpha - \delta\theta_x \cos\alpha \tag{56}$$

$$\delta\lambda = (\delta\theta_y \cos\alpha + \delta\theta_x \sin\alpha)\sec L \tag{57}$$

$$\delta h = \delta h \tag{58}$$

and the wander angle error equation is defined as:

$$\delta\alpha = \delta\theta_z - \delta\lambda \sin L \tag{59}$$

It should be noted that Equations (56) through (59) are only valid when not operating in the polar regions. The equations which define polar region behavior are very complex and are not considered in this analysis. This exclusion prevents the simulation of the INS in the polar regions although future implementation is an option.

To propagate the model it is important to consider the model in its generalized matrix differential equation form:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{F}(t)\delta\mathbf{x}(t) + \mathbf{G}(t)\mathbf{w}(t) \tag{60}$$

To deal with this model in a digital computer environment, the discrete form of Equation (60) is used. The solution of Equation (60) is [18]:

$$\delta\mathbf{x}(t_{i+1}) = \Phi\,(t_{i+1}, t_i)\delta\mathbf{x}(t_i) + \left[\int_{t_i}^{t_{i+1}} \Phi\,(t_{i+1}, \tau)\mathbf{G}(\tau)\,d\beta(\tau)\right] \tag{61}$$

which in turn can be written as a discrete linear stochastic difference equation [18]:

$$\delta \mathbf{x}(t_{i+1}) = \Phi\,(t_{i+1}, t_i)\delta \mathbf{x}(t_i) + \mathbf{w}_d(t_i) \tag{62}$$

The subscript d indicates a discrete time definition and the value $\mathbf{w}_d$ takes on the corresponding bracketed value from Equation (61) where $\mathbf{w}_d$ is the hypothetical derivative of the Brownian motion, β. The matrix, $\Phi(t,t_i)$, is the state transition matrix associated with the matrix $\mathbf{F}(t_i)$. It should be noted that Equations (60) through (62), in their most general form, would normally have a deterministic input term $\mathbf{Bu}(t)$. The simulation will not use any deterministic inputs, as they are not included in the model.

From this definition it can be seen that a solution to the INS truth model is possible by the evaluation of each term over a given computation interval. The main disadvantage of this approach is the computational burden of the evaluation of $\Phi(t_{i+1},\tau)$.

An alternate approach is to integrate the first term of Equation (60) and treat the noise term separately. The dynamics of Φ, in its linearized error state form, are slow enough that it can be assumed that, for the integration interval, Φ is constant. Taking advantage of this characteristic, by utilizing a numerical integration routine, and adding random noise $\mathbf{w}_d$ as in Equation (62) at the end of each computation interval, propagation of the model can be achieved.

The integration routine selected to propagate these equations must be both efficient and accurate. The fourth-order Runge-Kutta integration routine is considered to be a very capable integration routine. It can however, suffer from accuracy problems and does not have the efficiency to handle high speed computations accurately [24].

In discussion with Maybeck and after reviewing the techniques used in MSOFE, the technique chosen for this application is the Kutta-Merson technique [9,10,19].

The Kutta-Merson routine was evaluated against a number of other well known techniques to establish the best method for integrating trajectory type problems. The Kutta-Merson method proved to excel at handling trajectory driven systems such as the one in this thesis [7]. The Kutta-Merson equations are as follows:

$$y_1 = y_0 + \frac{1}{3}h * f(x_0,y_0) \tag{63}$$

$$y_2 = y_0 + \frac{1}{6}h * f(x_0,y_0) + \frac{1}{6}h * f(x_0 + \frac{1}{3}h,y_1) \tag{64}$$

$$y_3 = y_0 + \frac{1}{8}h * f(x_0,y_0) + \frac{3}{8}h * f(x_0 + \frac{1}{3}h,y_1) \tag{65}$$

$$y_4 = y_0 + \frac{1}{2}h * f(x_0,y_0) - \frac{3}{2}h * f(x_0 + \frac{1}{3}h,y_2) + 2h * f(x_0 + \frac{1}{2}h,y_3) \tag{66}$$

$$y_5 = y_0 + \frac{1}{6}h * f(x_0,y_0) + \frac{2}{3}h * f(x_0 + \frac{1}{2}h,y_3) + \frac{1}{6}h * f(x_0 + h,y_4) \tag{67}$$

where h is defined as the step size and y_n is the solution of the system equation at t_0 plus a step segment of h. The result of these equations is that the system is propagated from $y(t_0)$ to $y(t_0+h)$.

The technique is fifth order and incorporates automatic step control to optimize the integration. From the equations, the technique has two estimates of the solution at the end of the step, y_4 and y_5. These two estimates are used to evaluate step size validity and the step is adjusted to keep the error within the designated error tolerance.

The mechanics of this technique are discussed in detail in Chapter 3, Section 3.3.1.

The approach that is most commonly used to simulate the noise term w_d is to generate random noise in the form of the output of a random number generator with statistics $N[0,Q_d]$. For a discrete case, the statistics of w_d can be defined as [18]:

$$\begin{aligned} E\{w_d(t_i)\} &= 0 \\ E\{w_d(t_i)w_d^T(t_i)\} &= Q_d(t_i) \\ E\{w_d(t_i)w_d^T(t_i)\} &= 0, \quad t_i \neq t_j \end{aligned} \tag{68}$$

The quantity Q_d can be approximated by:

$$Q_d(t_{i-1}) = G(t_i)Q(t_i)G^T(t_i)\Delta t \tag{69}$$

As mentioned earlier, the slow dynamics of F(t) allows for the assumption that the dynamics are zero over the integration interval and therefore validates Equation (69) as an approximation to Q_d.

To achieve the necessary statistics for w_d, the numbers associated with the number generator must be translated into statistics $N[0,Q_d]$. This is normally achieved by taking advantage of a matrix square root algorithm such as the Cholesky square root [18]. Given a vector z of random number with statistics N[0,I], the necessary random noise can be achieved by Equation (70) [19].

$$w_d = \sqrt[c]{Q_d}\, z \tag{70}$$

In this thesis, however, Q in Equation (69), is diagonal and the square root algorithm is not necessary. Using the approximation of

Equation (69) to develop Q_d and since $G = I$, $Q_d = Q\Delta t$. From this, the value of white noise is equated as:

$$w_d = Q\Delta t z \tag{71}$$

2.2 Intermetrics INS Simulation Program

The Intermetrics program was developed to operate in the T-39 Simulation facility at Intermecrics Inc. The facility is developed based on standard form, fit, and function philosophy and is used to evaluate the components of navigation systems, specifically GPS at this time [6].

The program is a communication shell simulating INS message traffic based on the SNU 84-1 [4] and the MILSTD 1553B communication protocol [22]. Navigation simulation is limited to a constant heading and constant velocity vehicle trajectory about a spherical earth. Position about the sphere is resolved trigonometrically based on velocity and heading. No compensation is made for the "real" earth shape or for conditions such as the Coriolis effect and gravity. No consideration for navigation of a vehicle under acceleration is provided. Modelling and computation of INS errors are nonexistent.

The program is PC-based and written in C programming language. Some of the routines are also written in assembly language. It utilizes the PC1553-2 Ballard board [5] to interface to the 1553B bus. The simulation provides the capability to input two words, D01 and C01 of SNU 84-1, and read the output words I01, I06, I07, and I13. D01 is a generalized input to the INU from the control display unit and C01 is input from the CADC to the INU. I01 outputs the INU state vector, I06

and I07 provided output to the HSI/HUD/DISPLAYS, and I13 is an output initialization vector. The reader should refer to the SNU 84-1 specification [4] for details on the content of these words.

The program provides a display that indicates the simulation run-time, the INS mode, and the information contained in one of the input words and one of the output words. The displayed words are selectable from the keyboard. Present latitude and longitude are displayed and changed as the simulation progresses.

The program must read a data file which contains the velocity and heading of the vehicle in order to initiate the simulation in navigation mode. Velocity and heading are constant and only one input can be made per simulation cycle. The simulation has OFF, STANDBY, AIR ALIGN, STORED HEADING ALIGN, GYROCOMPASS ALIGN, OVERFLY, AUXILIARY, ORIENT, ATTITUDE, TEST, CALIBRATE, and NAV modes. Modes other than NAV are completed on a timing cycle and the appropriate bits are set in the output words. NAV mode is updated by the simple navigation technique mentioned previously. The simulation also has waypoint/markpoint capability. No emphasis is placed on this particular capability at this time.

The simulation acts as a remote terminal on the 1553B bus and the communication protocol is in the form of remote terminal to bus controller or bus controller to remote terminal transfers. MILSTD 1553B protocol is discussed in some detail in the next section. User interface with the simulation is through the terminal keyboard and through the bus controller. Once the simulation is started, all further communications is achieved through the 1553B data bus.

2.3 MILSTD 1553B Data Bus

MILSTD 1553 is a document which establishes a military standard for a communication system known as an Aircraft Internal Time Division Command/Response Multiplex Data Bus [22]. It is a 1 Mbps serial bus system which achieves avionics and stores management integration within an aircraft system. The standard provides methods of communication and the electrical interface for sub-systems connected to the data bus.

The key components of the system are the bus controller (BM), remote terminal (RT), bus monitor (BM), twisted shielded pair wire data bus, and coupling transformers. The bus controller, as its name implies, provides "absolute" control over transmission flow on the bus. It controls all communications on the bus and utilizes a command and response protocol method to communicate across the bus. The controller uses this method to control the system remote terminals, thereby controlling up to 32 sub-systems.

The remote terminal may be represented in two forms, the stand-alone concept, and the embedded concept. In the stand-alone concept, the RT is solely dedicated to the bus. Its only function is to communicate with the bus. It provides the interface between the controller and bus and a non-1553 system. System design may permit a stand-alone RT to also act as a bus controller. Note that a bus can only have one bus controller at a time.

In the embedded concept, the RT consists of interface circuitry embedded in the sensor or sub-system. It performs the data transfer in and out of the sensor but generally does not have controller capability.

This configuration is the typical approach in most Air Force equipment and is the case for the LN-94 INS.

The bus monitor monitors all messages and collects data for mass storage or remote telemetry. It may also serve as a back-up controller while observing the system state and operational mode.

The bus itself is composed of a twisted shielded pair wire which is coupled to the RT's and the bus controller by a bus coupler. The coupler provides isolation between the bus and the terminals either by direct coupling or transformer coupling.

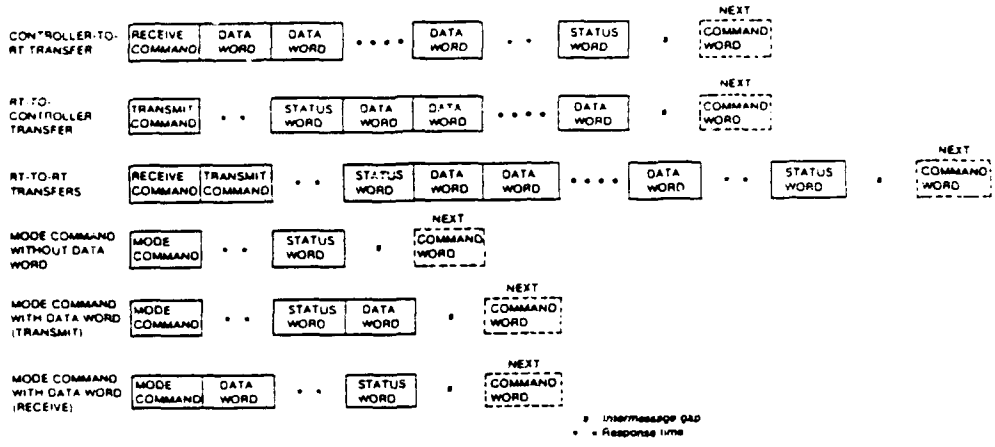


Figure 2.4 Information Transfer Formats

1553 protocol consists of 10 message types as shown in Figures 2.4 and 2.5. Within each message there are control words called command and status. Messages also contain data words which provide the necessary data to communicate on the bus. The messages are divided into two formats; information transfer formats and broadcast information transfer formats. The controller uses information formats to address individual

RT's, while broadcast messages address the entire bus or segments thereof.

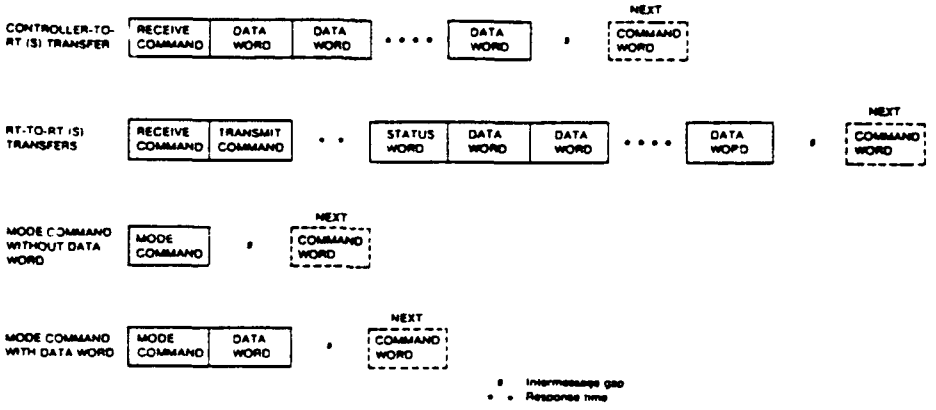


Figure 2.5 Broadcast Information Transfer Formats

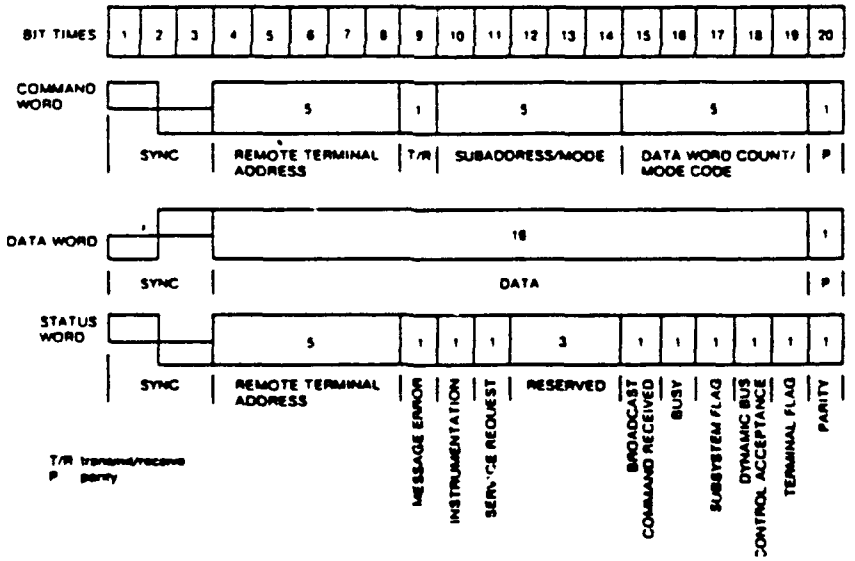


Figure 2.6 Mil Std 1553B Word Formats

In the information transfer format, each valid message sent must be acknowledged by an RT status wo d; hence, the command/response

protocol. Each word is 20 bits in length and consists of 3 synch bits, 16 data bits, and 1 parity bit. Details of the word structure are shown in Figure 2.6.

The command word can only be generated by a bus controller and provides to the RT the necessary information about the message to be transmitted. The command word address is the address of the RT that is to be controlled. The command word may be followed by another command word, by a data word(s), or a wait time to allow the RT to respond with a status word.

The status word, the response portion of the protocol, is utilized by the RT to inform the bus controller that a valid reception or transmission has taken place. Conversely, the bus controller utilizes the status word to monitor the health of RT's and the system as a whole. The RT sets bit flags in the status word which are then interpreted by the bus controller.

Transmission flow may be bus controller to RT, RT to bus controller, RT to RT, or as in the case of broadcast messages, bus controller to a number of RT's. Again, all transmissions are controlled by the controller using command words to address RT's and command functions.

As mentioned in Section 2.2, the INS simulation is an RT and in this case the address is 05. The bus controller is emulated by a DTI 1120 board [13] in another PC (see next section). The applicable words and formats for the 1553B bus for use in the simulation are provided in Appendix A.

2.4 Hardware

The simulation is written to operate under MS DOS using an Intel 80286 or 80386 processor. This particular installation is a 386-based 33 Mhz PC. The PC also has a Cybex 80C387 math co-processor to assist in the computational loading on the main processor. Data storage is provided in the form of a 120 Mbyte hard disk and a 1.2 Mbyte 5.25 floppy disk drive.

The 1553B data bus is achieved by using a Ballard Technology PC1553-2 interface board with the associated driver software [5]. The 1553B bus controller is emulated by a PC operating a Digital Technology DTI 1120 board with the SCE4 application software [13]. The bus is cabled between the DTI 1120 board and the PC1553-2 interface board.

2.5 Software

A number of commercial software packages are used to develop the simulation. They are as follows:

1. Microsoft C Compiler Version 6.0 [20]
2. Microsoft Macro Assembler Version 5.1 [21]
3. Essential Software C Utility Library Version 3.0 [11]
4. Ballard Technology Microsoft C Linkable PC1553 Driver [5]
5. Digital Technology SCE4 Application Software for the DTI 1120 interface board [13]
6. Companion disk software for reference [24]

In addition, INERCA [15], a trajectory generator developed by the US Air Force, was used to develop the vehicle trajectories for the simulation.

2.6 Summary

This chapter presents the theory on which the simulation is based. The LN-94 is a ring laser gyro strapdown INS which is implemented as a locally level wander azimuth mechanization. The original 93-state truth model is reduced to 23-states and the model differential equations are defined by Equation (21) through to Equation (52). Table 2.2 summarizes the states used in the reduced model. The model is linearized and represented by error states rather than whole states. This approach reduces the computation workload of the simulation.

The INS indicated values are developed as the sum of true trajectory and INS error. The true trajectory is provided by the INERCA trajectory generator and the errors are developed through integration of the error model using the fifth-order Kutta-Merson integration routine. The stochastic behavior of the model is provided in the form of a random number generator with output statistics $N[0,Q_d]$ where Q_d is approximated by Equation (69).

The Intermetrics Simulation [1] forms the groundwork for the simulation. However, it should be noted that the simulation developed in this thesis bares little resemblance to this program as a consequence of the additions/changes that are implemented.

The simulation operates under MS DOS on an IBM compatible 80386 33-Mhz PC with a Cybex math coprocessor. The simulation communicates through the MILSTD 1553 communication bus and provides information to the user based on the FNU 85-1 Specification [3].

III. Program Development

The executable program, INS_LN94, consists of 35 functions, not including MAIN, and 10 commercially developed functions, all encompassed in the INS_LN94 source code and the five linked object modules. This seems complex, but a close look at the logic flow and control provides a good understanding of the program.

This chapter provides an overview of the program, including logic flow and an explanation of the MAIN function. This chapter includes a discussion of the propagation technique used to process the simulation. Also provided are some insights as to the design philosophies for the program and considerations that were kept in mind when the program was being developed. The development tools are identified and reasons for their use are outlined.

The intent of this chapter is to show the global picture of the program. The reader is referred to Appendix E for a description of each function and to the source code for detailed coding information. Appendix F provides the necessary information on preparation and operating procedures.

As clarification to notation, all program variables and language syntax are capitalized to differentiate from the text. When referring to INS_LN94, this implies the entire executable program which includes the linked modules.

Specifically, this chapter is segmented into the following subjects:

1. Program Overview
2. Function MAIN
3. Model Propagation Method
4. Information Storage
5. Programming Considerations
6. Development Problems and Tradeoffs
7. Trajectory Generation

3.1 Program Operation Overview

INS_LN94 is a simulation of an LN-94 Litton Systems inertial navigation system. Through a simple question and answer interactive shell, the user can select a flight trajectory, a navigation session length, and storage of specified data. The simulation incorporates a 23-state error model and stochastic white noise injection to emulate the LN-94 INS. A Kutta-Merson fifth-order integration routine is used to integrate the system forward in time. The simulation responds to a MILSTD 1553 bus controller and provides two output messages to the bus.

The INS_LN94 is essentially a two-part program. It has an operating shell which controls entry into and out of the simulation, provides for parameter initialization, and provides plotting capabilities so the user may analyze data that has been generated. Within the control shell is the simulation process itself, which is in the form of the model propagation and input/output processes.

The program begins with a narrative overview of the simulation and provides details on operation procedures to the user. The simulation continues with a number of questions to the user, the answers to which

declare parameters to control the overall operation of the simulation. Following these questions, the simulation sets the video screen to the simulation format and simulation commences.

From the questions provided to the user, a number of choices are available. The simulation permits the use of three possible vehicle trajectories: static navigation, straight and level trajectory, and the Litton fighter trajectory. The simulation permits varying navigation session lengths up to the limits of the particular trajectory. The fighter trajectory is limited to a two hour flight, the static navigation has no time limit and the straight and level flight has a six hour time limit.

The user can store data to files. A choice of error states or whole states is provided. Error state collection is limited to 14 states from the error model. The whole states are the INS-indicated values provided by the simulation and again the limitation of 14 files of data is imposed. Refer to Section 3.4 for a complete list of the whole states and error states that are tracked and stored to file. In conjunction with data collection, the user can define the data collection rate by indicating the number of seconds between collection points. This feature gives the user flexibility with respect to storage space, an important option depending on the system in use.

On entry into the simulation, the user must align the system. The simulation will not enter navigation mode until the system is aligned. Alignment is simulated by a timing cycle. No "real" alignment takes place but the simulation indicates and follows the steps necessary to display alignment status. A four minute alignment takes place and the

system state vector is initialized with typical four minute alignment values. These alignment values are fixed for all simulations.

The user can select modes of operation with the defined function keys or exit the simulation by using the escape key. The user may monitor one of two output messages and the input message. The two output messages contain the navigation parameters of interest and are in response to command words #1 and #24 of the FNU 85-1 specification. The input message provides present position latitude and longitude and is formatted in accordance with and in response to command word #2. The user must initialize present position latitude and longitude prior to starting the navigation session.

The user is provided with the simulation time to monitor progress of the simulation. Only stationary navigation does computations in real time. Due to computational overhead, the other trajectories are not done in real time. Refer to Section 3.6 for more detail on this issue.

After the simulated flight the user has the option of plotting stored data to the screen or to a printer using a routine called EZPLOT. It is a simplified plotting utility made for the USAF. Other options include exiting the program or restarting the simulation.

Figure 3.1 shows the program hierarchy of the executable program. All source is written in C programming language except CLKT10.OBJ and INS_INT2.OBJ, which are written in assembly language. Assembly language is used in these cases to provide easy access to the machine level operations of the PC and the Ballard board. The function MAIN is part of INS_LN94.C. The five object modules contain functions that are called by MAIN. The Microsoft library [20] is included as part of the

compiler used to develop the program and the Essential library [11] is linked as an additional library. MCDRVL.OBJ is the object module provided by Ballard Technology [5] to operate the PC1553 interface board. Further details on program structure and compilation are provided in Section 3.5.

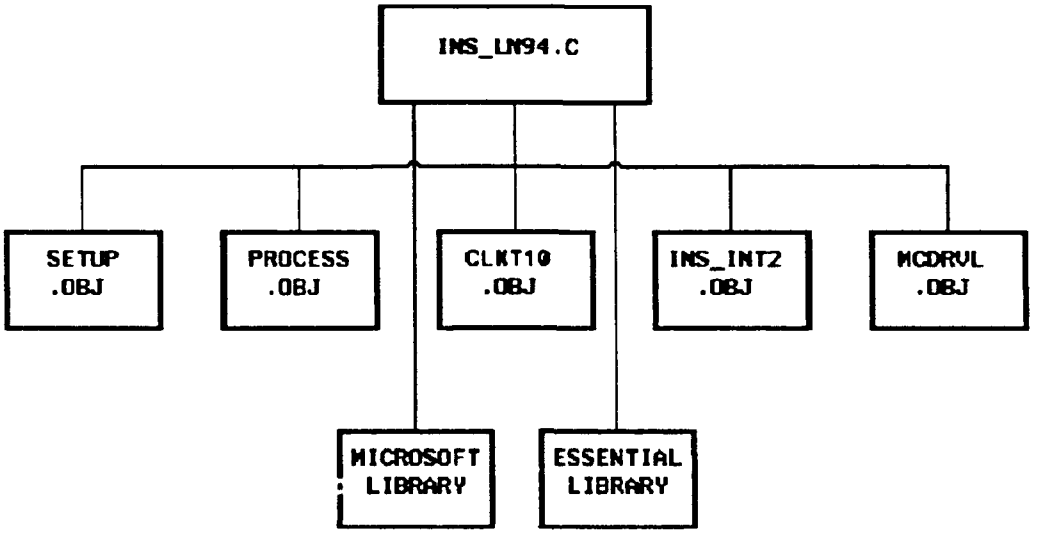


Figure 3.1 INS_LN94 Program Architecture

3.2 Function MAIN

Like all C programs, the program global control is developed and contained in the function MAIN. As can be seen from Figure 3.1, MAIN also provides control over the five object modules. To understand how MAIN provides this control, a flow chart is provided in Appendix D, Figure D.1 through Figure D.4. Reference to these charts as the discussion progresses aids in understanding the logic flow.

Before executing the simulation, MAIN asserts TEST_1553 which tests for the presence of a Ballard PC1553 interface board. Without the board, the simulation cannot be implemented on the 1553 bus. Once it

has been determined that the computer has the appropriate hardware, the simulation setup and processing begins.

The MAIN function has a number of decision processes that control the operation of the program. Examining the flow charts it can be seen that there are really only three major control points in the function. The first control, working from the outside loop to inside is, WHILE !SIMULATION_OFF, which maintains control over the entire program. The second control is WHILE EXIT_FLAG which maintains control over the simulation processing operation. The third control is an if statement, IF (HUNDRED_MS()), which controls timing and propagation of the model itself. Within IF (HUNDRED_MS()) there are a number of control statements. They rely on timing for operation and are therefore considered secondary.

Figure D.1 shows that WHILE !SIMULATION_OFF starts by introducing the program with INTRODUCTION and performs all the initializations prior to entering the actual simulation using INITIALIZE and DISPLAY_SCREEN. Control is then passed to WHILE EXIT_FLAG for the simulation process. As shown in Figure D.4, once the simulation process has terminated, the program enters PLOT_RESULTS, which gives the user the capability to plot simulation results or use the SIMULATION_OFF variable to terminate the program. INTRODUCTION also provides the user an opportunity to terminate the program prior to entering the simulation process. These are the only two areas where SIMULATION_OFF is accessed. Anytime the variable SIMULATION_OFF is TRUE, the program will terminate.

The second control, nested in WHILE !SIMULATION_OFF, is WHILE EXIT_FLAG, which maintains controls of the simulation process. This

loop takes control of the simulation once the simulation screen has been displayed. The main purpose of this loop is to give the user the capability to exit from the display screen and simulation process in an orderly fashion. Anytime the user wishes to exit the simulation, the ESC key can be input from the keyboard and the program performs an orderly exit from the simulation. The loop starts by initializing the simulation by reading data from the 1553 bus and setting the present position in the simulation with Process_CMD_2. The third control loop is then encountered.

This third control provides the timing of the simulation and stipulates when to call the necessary routines to propagate the system model. HUNDRED_MS tracks the passing of each 10 ms time interval. This is achieved by first setting the timer ENBTIM and new interrupt handler in INITIALIZE. This is achieved by replacing DOS interrupt 0 with a new interrupt handler and decreasing DOS timer 0 to call the interrupt handler every 10 ms. When the interrupt is called, a variable is incremented and this is tracked by HUNDRED_MS and the function indicates the passing of each 100 ms. The DOS interrupt handler continues to be called at 18.2 Hz to maintain computer system integrity.

The main disadvantage of this approach is that certain DOS functions will be affected by the changes made to the DOS timer and interrupt procedures. For example, the file date/time stamp will be incorrect. The advantage of this approach is that timing may now be set up to process in multiples of 100 ms, a simple number to manipulate. Using this technique, the propagation functions are started. See Appendix E for detail on these functions.

To make the simulation a real-time process, the computations to go from t_1 to t_2 must be faster than the actual time. Through some experimentation, it was determined that processing every second provided a viable time frame for processing as well as a reasonable time frame between screen updates and writing to the 1553 bus. Any shorter time interval tends to increase the computation overhead, not so much in the integration but in the process to initiate and return from the propagation function. Any longer time starts to introduce timing complications for other functions in the simulation. Accordingly, at each second of expired time, the simulation will enter the propagation cycle.

To control the length of time the system propagates, and to ensure propagation does not take place until the system is aligned, the following if statement is used:

```
IF (OP_MODE == NAV && NAV_RDY &&
                ((TIME(&CRRT_TIME) - ST_TIME_NAV) < SESSION_LENGTH))
```

ST_TIME_NAV is set when the simulation enters NAV mode and this is compared to system time. As long as the result is less than SESSION_LENGTH, the propagation cycle can be started. If all other conditions are met, the simulation begins to propagate. As long as the session length has not expired or the user has not selected the ESC key, the simulation will continue to propagate. The cycle, once entered, propagates until the one-second propagation interval has been completed. The key to real-time processing is that this interval must be completed in less than one second in order to return, process all other functions, and be prepared to propagate again at the beginning of the next second.

Following PROPAGATE, the simulation enters the function INS_ERROR_INCLUDE. This function processes the errors and trajectory data into whole state values. These values are then converted to the format necessary to output on the 1553 bus. Storage of whole and error states is also performed in this function.

In the same time period and following propagation, the functions PROCESS_CMD_1 and PROCESS_CMD_24 are called to calculate the INS-indicated measurements and to format the data to be sent to the 1553 bus. UPDATE is also called to update the processed information to the screen.

Every second as well, but out of synch from propagation by 300 ms, is the function Msg_Cmd_Proc. This function processes any commands that are input from the keyboard. The function is set out of synch with the propagation process to lessen the computational burden over that time frame and as a convenience so that the propagation functions can be controlled more precisely in their own control process. The final function that is called in this timing structure is WRITERTDWD, which is a 1553 function to pass processed data to the 1553 bus.

As the program steps out of the timing structure, a check is made as to the status of the navigation session. If the session is ended by elapsed time, IF FINISH_SESSION prepares the simulation to exit. A message is displayed to the screen to indicate the end of the session. The EXIT_FLAG variable is set to TRUE and the simulation exits to WHILE !SIMULATION_OFF control.

Once back into the main loop control, the timer and interrupt handler are reset to system values using REM_HDLR and DISTIM. Open

files are closed and the routine enters PLOT_RESULTS. Once in PLOT_RESULTS, the user may select to plot results, to restart the simulation, or to exit the program. The program calls a routine called EZPLOT to do any plotting requirements. Once plotting is complete and the user wishes to exit, the program processes the library function EXIT to monitor the program status at termination and to perform an orderly closure of the program.

3.3 Method of Model Propagation

Propagation of the system error model is controlled from within the function PROPAGATE. This section outlines the PROPAGATE process and shows, through the use of flow charts, the integration technique used to propagate the model.

The PROPAGATE function is called within the timing structure at one-second intervals and only when the simulation has been aligned, the operational mode is NAV, and the navigation session has not expired. Figure D.5 shows the logic flow of the function. PROPAGATE is directly responsible for the integration of the error model. In addition, it provides the noise injection capabilities that are necessary to emulate the real INS properly.

PROPAGATE calls two other functions within its process. The first, INTEGRATE, is the integration technique used in the simulation. Chapter 2, Section 2.1.4 outlines the philosophy for selecting this fifth-order Kutta-Merson routine. It is a variation of the FORTRAN routine that is used in MSOFE [9,10]. Changes were made to accommodate

the routine in C language and to use it specifically for this simulation. Section 3.3.1 provides more detail.

The second function, NOISE, is a variation of the function found in [24] and provides the randomly generated noise strengths with statistics $N[0,Q_d]$, which represents the stochastic white Gaussian noise that is required to implement the system uncertainties during propagation. Refer to Appendix E for more detail.

As discussed in the previous section, PROPAGATE is called from within the timing structure to propagate the system model over a specified time frame. PROPAGATE needs the value of the end time which represents the time to which the function will propagate before returning to MAIN. End time is incremented one second and passed to PROPAGATE. Recall that the one-second interval discussed in the last section represents the optimum processing time to maintain real time. Once end time has been passed to PROPAGATE, the process of integration over a one second interval begins.

PROPAGATE is controlled in a similar fashion as MAIN. It is encompassed by a decision loop, WHILE !FINISH. The purpose of this loop is to continue the integration function until end time has been reached. Depending on the trajectory, this process can be achieved in a one-second step, or as the trajectory becomes more dynamic, the one-second interval may be segmented into a number of smaller steps. The loop also gives the capability to return from the function INTEGRATE to add noise at each step interval.

End time is passed and INTEGRATE is called as the first step in PROPAGATE. On return from INTEGRATE, the status of the integration is

checked to ensure no errors occurred prior to continuing. If an error has occurred, the simulation terminates and an error message is displayed to the screen. The next step is to check the state that the propagation is in. If at a step interval or at the end of the integration interval, the function computes the Δt which represents the difference in time between the present time and the last time that Q_d was computed. NOISE is then called and the result is added to the solution vector from the integration function. The final step is to check if time has reached end time. If this is true, the variable FINISH is set to TRUE and the function exits the loop and returns to MAIN. Else the function steps through the propagation process again.

3.3.1 Function INTEGRATE. This function is the most complicated of all the program functions. It represents the basis of the computations used to propagate the error model. Figure D.6 through Figure D.8 show the logic flow of the function. The flow chart and the following discussion are limited to general logic flow and not a variable-by-variable analysis. The intent is to give the reader an understanding of the approach and not the exact programming techniques.

As indicated earlier, the integration technique used is a fifth-order Kutta-Merson process. Automatic step sizing is incorporated into the routine. By using the variables assigned in INITIALIZE, the integration routine may provide continuous noise injection or noise at the end time. In addition, the routine can be set to have a fixed step size or may enter an error checking process and adjust the step size automatically. The user also provides a maximum and minimum step size

and the error tolerance for the error checking process. These variables are found in INITIALIZE as well.

Once control is passed to INTEGRATE, the first operation that is performed is to initialize variables and check to ensure the parameters are within the range expected for the integration routine. This process is only done once at the beginning of the simulation. Comparisons are then made between the time interval of integration and the computer epsilon to determine if the interval is within the resolution of the machine floating point operations. Computer epsilon is a measure of the resolution of floating point number for the computer being used to execute the program.

If the interval is too small, the integration is done using the Euler method and results are returned to PROPAGATE. This is necessary since the Kutta-Merson technique further divides the time interval, which will then be out of range of the computer floating point operations and return erroneous values. Note that epsilon for a PC has been independently computed and is a defined constant in the program for double and single precision operations.

If the interval of integration is large enough to permit step sizing, the step size parameters are set and then checked to ensure the integration time + step size do not exceed the end time. If the step is too large, the step is reduced to the integration interval and integration continues. Step size segments are evaluated and then the step size is checked using a similar computer epsilon comparison as previously discussed. If the step is too small and if at the end, use

The function concludes by checking the state of the integration. If at end time, the function terminates and returns status to PROPAGATE. If the integration is set to provide noise injection at each step and the integration is at step time, the function returns to PROPAGATE for noise injection. Otherwise the function returns to Step to continue the integration until either end time or step time has been reached.

3.4 Information Storage

The process of storing data to files for later analysis is an important feature of the simulation. As indicated earlier in this chapter, the user is presented with options with respect to storage of data. The user can select whole state or error state storage, or select at what rate the storage will take place. Unfortunately, because the simulation executes on a PC, limitations are placed on storage.

The most limiting aspect is that DOS allows only a maximum of 20 files to be open at any one time. This doesn't seem too restrictive until one realizes that DOS uses five of those files for the operating system and the simulation uses one for the trajectory data. This leaves only 14 files for storage of data. Ideally, it is desirable to store all the states the simulation produces. Because of file limitation, only the more "useful" states are stored to files.

An option that was pursued to minimize this problem was to store multiple states in one file. This was not a viable option because it was considered that this would make the plots too busy and lead to less useful plots.

When the user selects the states to store, a preselected list of states is stored to disk. Refer to Table 3.1 for the applicable variables. The file names under which the states have been stored have also been included in Table 3.1

The other limiting aspect is storage capacity. Unlike the "virtually" limitless storage of a mainframe, a PC has some more realistic numbers for storage. This limitation does not present a great problem as long as the user is aware of his storage capacity and limits the data collection rate accordingly. As a guideline for storage, a three-hour simulation using 30-second collection intervals, occupies approximately 80K bytes of space

The storage process is "hardwired" in the simulation. If the user desires more flexibility in storage of data, the source code has to be changed and recompiled. However, the procedure as set down should handle most situations.

3.5 Programming Considerations

In deciding how to develop this simulation, a number of questions needed to be answered. Foremost was the programming language to be used and the system under which the program was going to be developed. As programming had already been started (Intermetrics simulation), the obvious choice was to continue with their process.

The selection of C as the primary programming language initially did not appear to be the best choice for this simulation. The original simulation was not very computationally intensive. With the introduction of the propagation cycle, the number of computations

Table 3.1 States Stored to File

Error States	File Name	Whole States	File Name
δL	del_lat.dat	longitude	long.dat
δl	del_long.dat	latitude	lat.dat
$\delta\theta_z$	d_thetaZ.dat	wander angle	wander.dat
ϕ_x	phi_X.dat	altitude	alt.dat
ϕ_y	phi_Y.dat	north velocity	vel_n.dat
ϕ_z	phi_Z.dat	east velocity	vel_e.dat
δV_x	vel_N.dat	vertical velocity	vel_v.dat
δV_y	vel_E.dat	north acceleration	acc_n.dat
δV_z	vel_Z.dat	east acceleration	acc_e.dat
δh	delta_h.dat	vertical acceleration	acc_v.dat
δh_c	del_h_c.dat	roll	roll.dat
		pitch	pitch.dat
		platform azimuth	azimuth.dat

increased 100-fold. Traditionally FORTRAN is a better "number cruncher" than other languages. However, the simulation needed some bit manipulation techniques as well. C is far better suited for this function. In addition, the simulation needed a good user interface and display screen manipulation. Again, C is better suited for these capabilities. The simulation is developed to operate on a PC and C is quickly becoming the industry standard for PC programming. As C is

reasonably effective in the area of floating point computations, it therefore appeared that C was the best overall choice.

C is not without disadvantages. One of its biggest disadvantages is readability. With all its shorthand coding techniques and the use of pointers, the program can become very unreadable to a programmer unfamiliar with C. To avoid this problem, the programs are well commented, including logical spacing of program statements. Another area of concern, to the structured programmer especially, is C's capability to be unstructured, namely, the promotion and demotion protocols for variables. C allows one variable type to be equal to another variable type without producing a compilation error. This allows the program to demote or promote a variable deliberately. Care must be taken not to misuse this technique as this can result in data corruption and loss of data resolution. This capability has proven to be valuable to the simulation program.

The previous program was written in Microsoft C. Microsoft's latest version of their optimizing C compiler, Version 6.0 [20], has been integrated to include an editor, system BUILD, and the Codeview debugger. Microsoft's MAKE utility has also been made more industry-compatible. It appeared to be the best selection and through the development has proven to be so. Version 6.0 also is closer to ANSI C and assists in maintaining portability. It should be noted though that Microsoft C still has some non-standard constructs that make the program difficult to move to other systems. In the simulation, few of these constructs are used and small changes should make this simulation very portable, although this was not one of the goals of the development.

The Intermetrics program [1] was written in Microsoft C Version 5.1 and some incompatibilities were encountered. Small changes in the code that was borrowed from this earlier program solved these issues with little complication.

The only disadvantage of utilizing the Microsoft C compiler is the lack of a DOS extender to take full advantage of the 80386 protected mode of operation with the associated 32-bit addressing and processing. The in-house WATCOM-386 compiler [28] was an unacceptable choice for compiler, as problems were being experienced with operating programs in protected mode. In addition, some of the library functions and syntax in the Intermetrics simulation are incompatible with WATCOM and present difficulties in utilizing any of its code. The goal was to develop a simulation and not be challenged with a finicky compiler. The only option left was to select the 286 version of Microsoft C and take advantage of the compiler in other aspects such as optimization. The downfall of this choice is the loss of computation speed from not fully using the capabilities of the 80386 process.

The simulation is developed using the compiler's small memory model. The small model is most applicable in this case as the program code and data segments do not exceed 64K each. More importantly, the small model only uses direct addressing where, the default for variables is NEAR. This method of addressing provides the fastest code.

One of the reasons that the Microsoft C compiler was selected was its capability to optimize the source code. All C language source was compiled with the optimizations listed in Table 3.2.

Other compiler options include compiling under the 80286 processor and utilizing in-line 80287 coprocessor commands. For this version of

Table 3.2 Compiler Optimizations

Code	Optimization
Ot	enables optimization of code for speed of execution
Oi	enables optimization of function execution allows intrinsic form of function to be used
Ol	enables optimization of loop code replaces loop code with more efficient code
Gs	removes stack checking code to produce faster executing code

Microsoft C, the code only utilizes 16-bit words. This obviously means that the program is not taking advantage of the 80386 32-bit word processing. The best this simulation can do is take advantage of the 386 machine CPU speed. In-line coprocessor commands are the fastest option available for coprocessors. The only disadvantage of this procedure is that, if the machine used to execute the simulation does not have a coprocessor, the simulation performance is degraded. Other options allow the flexibility of switching between machines with and without coprocessors. These option aren't quite as efficient and since this program requires a coprocessor, dedicated coprocessor compilation is used.

The linking process involves using the compiler-integrated BUILD process. All linkable modules are listed in a program list and the list is processed as a function of the host source code, in this case INS_LN94.C. Link options include command exepack, which removes sequences of repeated bytes in the executable code. This optimization decreases the size of code and reduces the program load time. The linker also establishes a stack size of 7K. Stacks of this size are not normally used; however, in this case a large number of global variables and the recursion in parts of the program necessitate a relatively large stack.

Chapter 2, Section 2.5 provides a list of the other software tools used in the simulation development. The assembly language programs are assembled to object code by Microsoft Macro Assembler Version 5.1 [21]. The C Utility Library [11] is linked with INS_LN94.C as an additional library.

3.6 Development Problems and Tradeoffs

It is well known that software development is a time consuming process. This simulation is no different. The development stage took a disproportionate amount of research time. It is also well known that a program is never at its best level of development. It can be subject to constant changes and updates as the software is utilized. One could spend the entire research time in development alone.

To avoid this programming pitfall, and to leave sufficient time for validation, a number of tradeoffs were accepted for problems encountered in the development. The first problem was in the trajectory

generation and data manipulation process. To make the best use of the data, a spline-fitting algorithm is used to produce data in between the points that are generated by the trajectory generator. The algorithm proved to be unworkable in the time frame provided. The alternative that has been implemented is to assume that, for a short interval, the trajectory can be considered constant (the output of a time-invariant model), and the model is integrated over this time interval. To do this requires large trajectory data files with data provided at one second intervals. In addition, the trajectory must be benign enough to allow this assumption to hold.

Problems were also encountered in reading the unformatted data from the trajectory generators. The alternate method is that INERCA [15] provides a report file which is an ASCII file of the trajectory data. This is dismantled by a conversion program to generate the appropriate data. This alternative approach excludes the use of PROFGEN [2] as a trajectory generator. Add to this the limitation that INERCA cannot provide roll, pitch, or yaw trajectory data.

The consequence of these problems and resulting tradeoffs is that the Litton fighter profile is not implemented. The interface for the profile is maintained in the program as described up to this point, and all the conditions of operation still remain valid. This trajectory is unselectable until further development is performed. Chapter 4, the simulation validation, does not address the fighter profile at all.

The last problem in the development stage is the concept of real-time processing. The simulation processes the static trajectory in real time while the straight-and-level trajectory operates almost in real

time, losing approximately two seconds for every 30 minutes of operation. Further development of the propagation module in terms of efficiency and/or assembly language programming should solve this problem.

It should be noted, for clarification, that the real-time losses are a measure of the difference in the simulation clock as compared to the system clock that is used for debugging the simulation and measuring real time. The simulation clock is dependent on program performance to maintain the simulation time as close to the system clock as possible. The system clock is developed directly form the timing pulses of the computer clock. Refer to the discussion of function Update, in Appendix E, for details on this system clock.

3.7 Trajectory Generation

As a result of the problems outlined in the previous section, only one flight trajectory is generated for the simulation. The profile is referred to as a straight-and-level profile. In fact this name is slightly misleading in that the trajectory follows a great circle route. This allows changes in navigation parameters without being too dynamic. A constant altitude of 5000 ft and a constant velocity of 586 ft/sec are maintained for the duration of the flight. The trajectory starts at N 45 degrees latitude and W 84 degrees longitude. The flight is generated for six hours of operation, although for validation purposes, the flight is only two hours in duration, as the accuracy specifications are for two-hour flights.

The trajectory data is stored in file, straight.trj. This file must be stored in the same directory that the simulation stores data so the simulation can access the data from them.

3.8 Summary

This chapter gives the reader some insight into the development of the simulation, INS_LN94. The discussion is limited to the two main processes in the program, function MAIN and function PROPAGATE. These functions are the key to the simulation. MAIN provides the simulation control while PROPAGATE is the function which controls the integration of the error model. Within PROPAGATE is the function INTEGRATE which is treated in some detail in Section 3.3.1 of this chapter. It provides the means of integrating the model forward in time. Appendix D provides flow charts of these functions.

The simulation was developed using the Microsoft C Optimizing Compiler Ver 6.0 [20] and the program is optimized using parameters listed in Table 3.2. The compiler does not take advantage of the 80386's 32-bit processing, but for reasons outlined in Section 3.5, this compiler proved to be the best choice.

Section 3.6 outlines a number of problems encountered during the development of the simulation. The problems impose the restriction that the fighter profile is not useable at this time. Further development should solve this problem.

Information may be stored for later analysis using the simulation interface to select the appropriate information. Table 3.1 outlines the information that may be stored. Whole states or error states may be

tracked and stored and a plotting routine called EZPLOT may be utilized to view data.

The reader is referred to Appendix E for a description of all the simulation functions and to Appendix F for simulation user operating procedures.

IV. Simulation Validation

The simulation is of little use if it is not validated to assess its accuracy in emulating the LN-94 INS. In this chapter the issues related to validating the model and the simulation operation are outlined. The validation process is completed in three steps. The first step is to analyze the performance of the 23-state model used in the simulation. There are certain assumptions made about the 23-state model, so this step in the validation process provides a check of the proper implementation of the model rather than the appropriateness of the model.

The second step is to compare the performance of the simulation against the outputs of an actual LN-94 INS in static navigation, and to simulate a flight. This step provides the most concrete measure of the fidelity of the simulation. This process shows how closely the simulation emulates the operation of an actual INS, and how well the simulation follows a true trajectory.

The final step is to check the performance of the noise generation process. This process involves analysis of the random number statistics and the general behavior of the noise produced by the stochastic white noise generator function NOISE. Refer to Appendix E details on this function.

This chapter explains the validation process as described above. Procedures are outlined and conditions of the validation are presented.

Data resulting from this process is presented in applicable Appendices, and a discussion of the results is provided.

4.1 Model Validation

As noted in Chapter 1, the purpose of this research is to implement a proven model of the LN-94 into the simulation. As such, this means finding a reduced-order model of the LN-94 that has undergone sufficient validation to be considered a valid model. Through consultation with various people doing on-going research in this field, specifically Stacey and Paschall [23,27], the decision was made to develop the simulation based on a 23-state model which had been validated by Stacey during his research [26]. Stacey's validation process included a comparison of a number of simulation runs on MSOFE [10] against the results of the full 93-state Litton model. It was decided that the fidelity of Stacey's model was sufficient to use in this simulation.

One type of validation necessary for this model is to ensure the appropriate behavior of the model under various conditions of gyro and accelerometer error for the static navigation case. This validation is necessary since the model is programmed in a different language and is executed within a different operating environment. This validation is also required to ensure the integration technique is functioning properly.

The baseline information used in this validation is the unified analysis done by Britting [8]. Britting analyzes the behavior of a 9-state INS error model when gyro drift or accelerometer bias is applied

to the north, east, or down axis. An exact correlation between Britting's results and the results of this model is not possible, but general characteristic behavior is identified. It is important to note that Britting's analysis only considers a two-accelerometer local level system while the Litton model is a three-accelerometer system. The result is no vertical channel components that affect the LN-94 model have any effect on the Britting model.

To perform this validation, the propagation module of the simulation is executed. The module is coded identically to the simulation propagation process except that there are no real-time considerations. The module tracks error states while integrating forward in time over a designated time interval. Noise is not introduced into the model during integration. This is to ensure the same conditions as the Britting analysis. The trajectory data used matches the Britting analysis with longitude set to zero degrees, the latitude set to 45 degrees, and the gravity acceleration set to one g. All other trajectory data values are set to zero. Data is collected every 30 seconds over a simulated 36 hour time frame. All initial conditions are zero. The bias or drift of interest is applied at a constant value of one meru (milli-earth-rate unit, .015 deg/hr) of gyro drift or 10^{-4} g for accelerometer bias. The analysis looks at latitude, longitude, and attitude errors for a given gyro drift or accelerometer bias.

Another difference between the Britting analysis and the LN-94 implementation is that the LN-94 coordinate system is implemented in the East-North-Up convention, while the Britting convention is North-East-

Down. The transformation between the LN-94 convention and the Britting convention is:

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} E \\ N \\ U \end{bmatrix} \tag{72}$$

This presents some complications when making comparisons. Without explicitly deriving Britting's equations in the ENU convention, it can be seen that this transformation changes some sign conventions in Britting's dynamic matrix and also switch sensitive axes uncertainties. This difference in convention means that for analysis ϕ_x equates to ϵ_E, ϕ_y to ϵ_N, and ϕ_z to $-\epsilon_D$.

Appendix G shows the results of the 23-state model validation. One of the obvious features of the simulation results is the magnitude of the error. In all cases except for ϕ_z, the magnitude of the error is on the order of 10 times greater than that found in the Britting results. This increase in magnitude has even elevated magnitudes of error in the azimuth gyro drift case to readable levels.

This difference is significant and would lead one to question the validity of this comparison. Keeping in mind the intent of this part of the validation process, and after examining the performance of the simulation in the other validation processes, this magnitude difference may be overlooked at this stage. Refer to Section 5.2 for further discussion of simulation validation.

It is likely that part of this difference is due to error model definition. If a detailed term-by-term comparison of each error equation is performed, though this is not done for this document, it

shows inclusion of terms in the LN-94 model that are not present in the Britting model. Most evident, as mentioned previously, is the inclusion of the vertical errors in the position and attitude equations of the LN-94.

From a modal point of view, simulation results are consistent with Britting results. For gyro drift, Foucault modulation occurs as a first order influence in level alignment errors, Figures G.6-G.7, G.11-G.12, and G.16-G.17. As expected, it is a second order influence on latitude, longitude, and azimuth errors, as seen in Figures G.4-G.5, G.8, G.10-G.11, G.13-G.15, and G.18. A Foucault frequency of 34 hours (consistent with 45 degrees latitude) is apparent, as is the 84-minute Schuler period. For the accelerometer bias plots, again the modes are consistent. The Schuler mode is dominant and the same 34-hour Foucault frequency is evident, as in Figures G.21-G.24 and G.26-G.29. Due to the type of implementation that Britting presents, the vertical accelerometer is not plotted.

In both the gyro drift case and the accelerometer bias case, ϕ_z does not respond in the same fashion as in the Britting analysis. Note that the error magnitude of ϕ_z for accelerometer bias for the simulation is considered to be zero, Figures G.25 and G.30. What is plotted is at the noise level and considered inconsequential.

Errors in the vertical axis are unstable. Again, this difference is directly related to the inclusion of the vertical channel. This difference is also attributed to the fact that the vertical axis is uncommanded, as the LN-94 is a wander azimuth implementation.

It is seen from the results that the error model exhibits the proper modal behavior. Performance of the 23-state model indicates with some confidence that the model and the associated C language coding performs well in the simulation. The integration technique performs as expected. Although this is not a complete validation, it does provide the necessary insight into the operation of the model and the integration technique to be able to proceed with the simulation validation.

4.2 Simulation Validation

To validate the LN-94 simulation, the ultimate test is a comparison of data generated by the simulation and data generated by an actual INS. For this validation, no actual flight data is available; however, a static navigation data set is generated and is used for this validation process. The validation also includes analysis of a simulated straight-and-level flight trajectory.

*4.2.1 **Accuracy Considerations***. To evaluate the performance of the simulation, accuracy benchmarks must be established. FNU 85-1 [3] specifies the accuracy requirements for the LN-94. Table 4.1 summarizes the accuracy limits of interest. These accuracies are based on a two-hour flight so any comparisons are made up to and including the two-hour point on plots.

Present position accuracy is measured in terms of CEP or circular error probable. CEP is a measure of the probability that the indicated present position will be within a specified radius of the true position. CEP is defined as:

Table 4.1 LN-94 Navigation Accuracy Limits

Navigation Parameter	Accuracy (4 min GC Align)
present position lat and long	0.6 nm/hr (cep) 2 hour flt
V_N, V_E, V_U	±2.5 ft/sec
wander angle	±0.033 deg (5.76e-4 rad)
pitch	±0.033 deg (5.76e-4 rad)
roll	±0.033 deg (5.76e-4 rad)
platform azimuth	±0.05 deg (8.73e-4 rad)
A_N, A_E, A_U	±0.5 ft/sec^2
altitude	150 ft

$$CEP = 0.83 \sqrt{\frac{\sum_{n=1}^{N} (RER_n)^2}{N}} \tag{73}$$

N is the number of missions and RER_n is the radial error rate on the n-th mission, which for terminal information, is the difference between the actual position and the indicated position for the period of the flight.

The measure of a good INS is its capability to track a true trajectory. In the case of a flight this is easy enough to measure as the difference between the true and the indicated trajectory. The actual INS static case is somewhat different. The actual environment is not reproduced exactly in the simulation. A model does not precisely follow the actual piece of equipment. Models are by definition approximations to some degree. Typically, high frequency influences are not taken into consideration and therefore do not impact on the model.

In addition, in the static navigation case is the fact that the specific force information is not be modelled in the INS truth model. It is required as an input. This is a problem if the INS environment is uncontrolled. Environmental vibrations are a minor concern in that they cannot be isolated and are not realistically simulated for this thesis. This results in unpredicted specific force influences on the INS.

As a result, there is no true trajectory for the INS other than the fact that it theoretically should be reporting its position as standing still. This makes accuracy measurement somewhat arbitrary. The accuracy benchmarks must be set by the developers. After some discussion and application of engineering judgment it is decided, for this study, that the static navigation simulation must maintain present position latitude and longitude within ±1% of the actual INS position and exhibit the same characteristic behavior. All other states of interest must exhibit the same behavior as their actual INS counterparts.

The criteria of Table 4.1 is used for both the simulated static navigation case and the simulated flight. For both cases, the accuracy criteria are applied with respect to the true trajectory.

4.2.2 Data Collection. Data for static navigation is collected from the LN-94 in the AFIT Navigation Lab. Using the 1553B bus controller, the INS is aligned at the AFIT latitude and longitude, N 39 degrees 46.925 minutes and W 84 degrees 4.992 minutes. After alignment, the INS is set to NAV mode for a three-hour navigation session. A three-hour collection is made to look at behavior beyond the usual two-

hour limit. Data is collected from the MILSTD 1553B bus every 30 seconds.

The method of collection is an in-house program executed on NAVSTAR, which samples the bus data and converts designated words of data into decimal format and stores them to file. A conversion program was written to make the data usable with the plotting routines used in this thesis.

A similar approach is taken with the simulation. The initial position is set by the MILSTD 1553B through the bus controller to the same position as the actual INS. The initial state vector is declared

Table 4.2 Initial State Vector

State	Value	State	Value
δx_1	0.0	δx_{13}	0.0
δx_2	0.0	δx_{14}	9.696391e-9
δx_3	0.0	δx_{15}	9.696391e-9
δx_4	1.084e-5 rad	δx_{16}	9.696391e-9
δx_5	1.084e-5 rad	δx_{17}	6.43972e-5 ft/s
δx_6	5.8178e-4 rad	δx_{18}	6.43972e-5 ft/s
δx_7	0.0	δx_{19}	6.43972e-5 ft/s
δx_8	0.0	δx_{20}	2.424071e-5
δx_9	0.0	δx_{21}	2.424071e-5
δx_{10}	0.0	δx_{22}	0.0
δx_{11}	0.0	δx_{23}	2.236 ft
δx_{12}	0.0		

as per Table 4.2. Selection of these initial conditions is a result of

examining the returns from an MSOFE simulated alignment [10,27]. The 1σ values provided in the Litton CDRL [17] do not necessarily represent the typical alignment values of the INS. It was necessary to examine the results of an INS alignment to ascertain the values one might expect to see after an INS has aligned itself.

A number of initial conditions were attempted, but they gave erratic behavior or large error propagation. The conditions outlined in Table 4.2 provided the best performance and are substantiated by the validation results. The initial conditions are coded into the simulation and for all cases the simulation starts with these values. The simulation does not actually perform an alignment. A timing loop is executed and these conditions are set.

The simulation software is initialized to inject noise into the model at the end of each step. The maximum step size is set to one second and the minimum step size is set to 10^{-20} seconds. Error tolerance is set to 10^{-4}. These parameters are not particularly critical for stationary navigation. The navigation is benign in nature and step size does not change and errors are minimal. These parameters become more important as the dynamics of the flight increase and become variable.

Trajectory data for the static case include present position latitude and longitude, vertical acceleration equal to one g, and a wander angle of -1.588054 radians. All other parameters are set to zero. The wander angle is included in the trajectory data, as it reflects the wander angle to which the actual INS aligns.

The simulation is set for a three-hour navigation session and NAV mode is selected. The data from the simulation is collected every 30 seconds. It should be noted that the data is collected from the solution of the integration routine just prior to formatting for the MILSTD 1553B. This differs from the data collection procedure for the actual INS. Data is collected directly from the 1553B bus. This difference is necessary to reduce the computational load on the simulation. Otherwise data would have to be converted back into decimal form prior to analysis.

The bus words have a limited resolution, while the simulation data is a double precision floating point value. This is worth noting but does not present any great difficulties. Data is collected for the parameters listed in Table 4.3. Plotted results of this data are shown in Appendix H and a summary of maximum error magnitudes is provided in Table 4.4.

Table 4.3 Navigation Parameters Collected for Comparison

Navigation Parameter	Navigation Parameter
present position lat and long	pitch
velocity	roll
altitude	platform azimuth
wander angle	acceleration

Data collected for the simulated flight is the same as for the static flight, with the exception that the trajectory data is provided by a data file and selected as described in Appendix F. Data is

collected every 30 seconds for a two-hour period. Plots of data collected are shown in Appendix I and are analyzed in detail in Section 4.2.4.

4.2.3 Static Navigation Performance Analysis. Two of the most important navigation parameters are latitude and longitude. The comparison of these parameters indicates close emulation of the LN-94 by the simulation. The error bounds of the simulation fall well within the one percent allowable. Calculations based on the two-hour indicated position reveal a change in position of 4071.17 feet or a rate of 0.34 nm/hr. This equates to a CEP of 0.27, well within the bounds of the FNU 85-1 specification of 2.6 for a single flight.

Table 4.4 Simulated Static Navigation--Summary of Maximum Error Magnitudes

Navigation Parameter	Maximum Error (2-hr flight)
present position	4071.17 ft (0.34 nm/hr)
altitude	150 ft
X velocity	0.52 ft/sec
Y velocity	1.4 ft/sec
Z velocity	1 ft/sec
wander angle	2.6×10^{-5} rad (0.0014 deg)
roll	0.0 rad
pitch	2×10^{-5} rad (0.0011 deg)
platform azimuth	5.0×10^{-4} rad (0.028 deg)

Figure H.3 depicts the altitude behavior of the simulation. A plot of the INS altitude is not provided, as it is zero. This is the case as the altitude update for the INS is not provided as a measure of

baro-altimeter uncertainty but is indicated as a true value at all times. There is no baro-altimeter model in the actual INS installation. What is shown in Figure H.3 is the effects of the simulated baro-altimeter uncertainty. The values of ±150 ft are quite reasonable for a barometric altimeter [3,23]. To achieve this result the noise component of δh_c was reduced from the model documentation value of 33.0 to 4.0.

Figure H.4 shows the wander angle behavior. The trend of the simulation is the same as that of the INS, and values again are well within the one percent allowable error. The simulated two-hour value of the error in wander angle is very small and is well within the accuracy bounds specified in Table 4.1.

Velocity behavior in the X axis is very close to the INS data. Velocity in the Y direction, Figure H.6, is the only state that behaves contrary to the actual INS. The difference error between the INS and simulation is quite large and the growth of the INS velocity is significant compared to the simulation though within limits of accuracy in Table 4.1. Some of the growth may be explained by the real environment of the INS versus simulated environment. Velocity in the Z direction is consistent with the baro-altimeter. Recall that, for the INS, there is no baro uncertainty. Accordingly one would expect the vertical velocity to be zero relative to the simulation data. Again, accuracy criterion is met.

Roll, pitch, and platform azimuth show equally good performance, although their emulation of INS data is not as close as the other parameters have been. Behavior is similar and, for the simulation, the FNU 85-1 accuracy is being met.

Figure H.11 and Figure H.12 are provided to highlight the previous discussion of vibration and its effects on the INS performance. The level accelerations are equal and non-zero. This is perceived as high frequency vibration of the INS. Heavy equipment next door to the lab is likely to be the source. Based on this, it is expected the actual INS errors will be greater in general than the simulation errors, and they are. It is important to keep in mind that the real world of the INS is very difficult to reproduce in the simulation.

Tuning of the simulation model proved to be relatively ineffective at making the simulation match the LN-94 simulation with the exception of the baro-altimeter uncertainty. By tuning the baro-altimeter uncertainty, typical baro-altimeter behavior of ±150 ft is achieved.

4.2.4 Flight Simulation. The simulated flight validation looks at the results from the point of view of the limits set in Table 4.1. If the simulation can follow the true trajectory within these limits, then the simulation is considered valid. The simulation is validated with a straight-and-level trajectory only.

The straight-and-level profile is along a great circle route to highlight some of the operating characteristics of the INS for teaching and research purposes. The setup and data collection is as defined previously. Appendix I provides results of the simulation compared to the true trajectory. Appendix J provides plots of the error magnitudes of the simulation as compared to the true trajectory and Table 4.5 shows these errors in tabular form.

Results of the validation are excellent. Figure J.1 and Figure J.2 provide the magnitude of error in present position. From

Table 4.5 Simulated Straight-and-Level Trajectory--Summary of Maximum Error Magnitudes

Navigation Parameter	Maximum Error (2-hr flight)
present position	5325.19 ft (0.44 nm/hr)
altitude	160 ft
north velocity	1.95 ft/sec
east velocity	0.65 ft/sec
vertical velocity	1.16 ft/sec
north acceleration	$6x10^{-5}$ ft/sec^2
east acceleration	$6x10^{-5}$ ft/sec^2
vertical acceleration	$6x10^{-5}$ ft/sec^2
wander angle	0.007 deg
roll	0.00104 deg
pitch	0.0022 deg
platform azimuth	0.0333 deg

these values, the indicated position after two hours is 5325.19 ft from the true position with an RER of 0.44 nm/hr. This provides a CEP of 0.37. Present position errors are well within the accuracy parameters of Table 4.1.

Altitude behaves as predicted in the static simulation case. The baro-altimeter uncertainty drives the altitude state to errors within the 150 ft accuracy limit. Velocities display the characteristic sinusoidal behavior and are limited to magnitudes of less than 2.0 ft/sec, as seen in Figure J.4 to Figure J.6.

The magnitudes of acceleration in the north and east directions, as depicted in Figure I.7 and Figure I.8, are very small. As a result, the initial conditions influence the error behavior more than expected.

Over time the initial condition influence is reduced and the simulation follows the true trajectory extremely closely. As acceleration magnitude increases to more typical levels as in the vertical acceleration, Figure I.9, the effects of initial conditions become insignificant.

Roll and pitch error behavior, as seen in Figures J.11 and J.12, are within accuracy limits. The only parameter that falls outside the accuracy limits is platform azimuth, portrayed in Figure J.13. The azimuth error starts outside the limits and over time starts to move towards the true trajectory. This leads one to believe that the initial conditions may be impacting on this parameter. Platform azimuth is a function of the solution to ϕ_z. The initial condition for this state is set well below the 1σ value designated in the Litton CDRL and is consistent with values for an INS alignment. The error in platform azimuth is just outside the limits of the FNU 85-1 specification (Table 4.1) and is considered reasonable without any further data for comparison.

In all cases, the simulation performs extremely well in emulating an INS and in staying within the accuracies expected of the LN-94. Very little tuning of the model was required to achieve this accuracy. A case can probably be made for increasing the values of the noise strength in the model to better emulate the "inaccuracies" of the LN-94. This would introduce higher levels of uncertainty into the noise process and induce greater magnitudes of error in the model. To do this realistically, flight data for the performance of the LN-94 would be required. States dealing with accelerometer and gyro uncertainties

would be ideal candidates for this treatment, but again with no baseline data for comparison, the attempt to emulate the exact accuracy of the LN-94 would be futile. The ability to stay within the LN-94 accuracies is sufficient substantiation to consider the simulation valid.

A case can also be made for increasing the number of states in the model. Rather than relying on noise parameters to produce the additional errors, some of the errors that are lost in the model reduction can be reintroduced to the model.

4.3 Noise Process Validation

The method of generating the Gaussian white noise which provides the randomness of the model process is a critical one. As is discussed in Chapter 2, the stochastic process must have statistics of $N[0,Q_d]$. The techniques used to develop these statistics are discussed in some detail. The one thing that remains to consider is whether the random number generator provided as part of the compiler can provide the random numbers with the appropriate statistics.

The number generator must be able to produce scalar numbers in a Gaussian distribution with statistics $N[0,1]$. The supplied generator provides uniform deviates between the values zero and one. The number generation process involves summing twelve random numbers and then subtracting the value 6.0 from the sum. The summing process provides approximately Gaussian characteristics and the subtraction sets the mean to zero.

Figure 4.1 shows the results of generating 2500 random numbers from the function Random_Number. The sample size is not significant. A

number of different sample sizes ranging from 50 to 5000 were evaluated. All produced the same kind of statistics. The typical Gaussian shape can be seen and the statistics for this sample are N[-0.011863,1.01145].

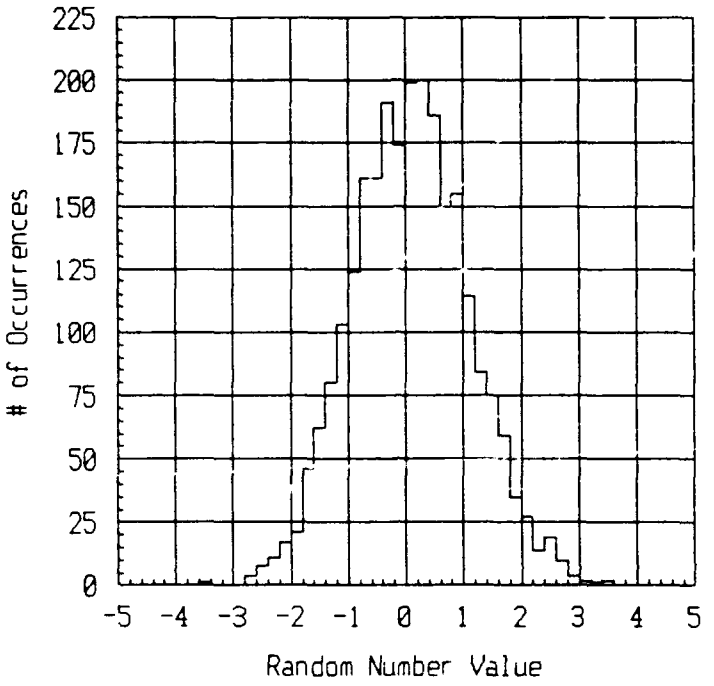


Figure 4.1 Random Number Statistics

The deviation from the zero mean value expected is acceptable, though further refinement could be achieved by compensating for this difference before the numbers are used in the noise process. Despite this difference in mean value, by utilizing numbers with these statistics, one can be confident in that equating Equation (71), the appropriate stochastic noise can be generated.

Figure 4.2 shows an example of the noise generated by the simulation for state four of the 23 state model [4]. The randomness of

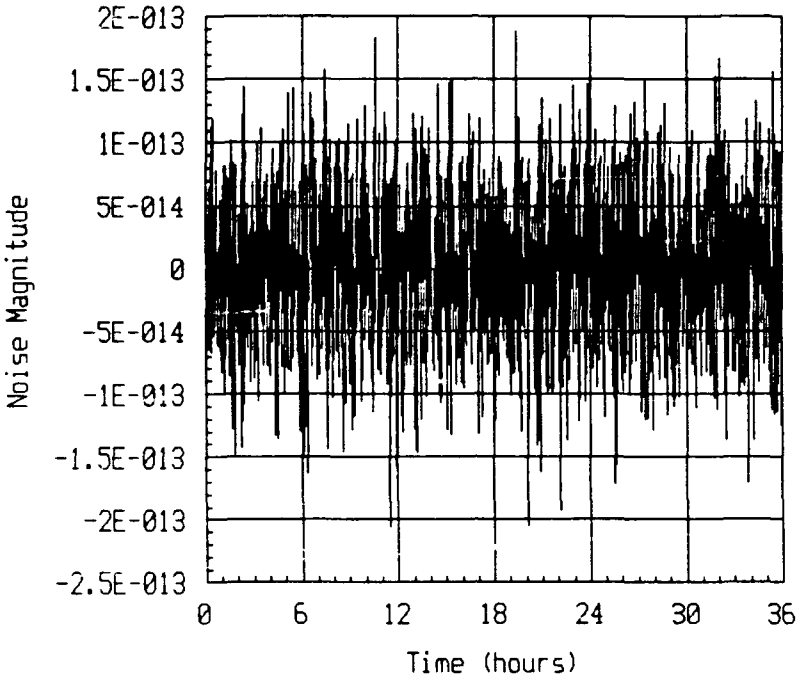


Figure 4.2 Noise Characteristics of State Four

the noise is noted in the noise behavior and the approximately zero mean value is apparent. The magnitude of the noise is consistent with the values expected for the model. The selection of the state is not important as analysis has shown that each noise source is behaving in comparable fashion.

4.4 *Summary*

This chapter discussed the validation of the simulation. The results indicate that the simulation is emulating the actual INS performance quite well. The criteria, to maintain less than one percent difference error between the INS and the simulation, is met in the case

of static navigation. Environmental influences and higher order errors can account for differences between the two sets of data. The implementation of a reduced order model in the simulation can account for some of these differences as well. Some error emulation capability is lost in the reduction. Appendix H summarizes the static navigation simulation as compared to actual INS static navigation data. Table 4.4 displays maximum errors produced by the simulation for the static navigation case.

Appendix I and Appendix J summarize the results of the straight-and-level flight trajectory. Table 4.5 summarizes the maximum errors encountered in the simulation. The accuracy of the simulation can only be measured as a function of the accuracy limits in Table 4.1. As is seen by the data presented, the simulation does meet this benchmark. Unfortunately, without flight data from the actual INS, tuning the simulation, as indicated in Section 4.2.4, is not possible. Validation shows that the simulation is a good emulation of the LN-94 based on benchmarks that are available.

The noise generation technique is discussed to validate its performance. Figure 4.1 indicates that the statistics of the random number generator are as required and Figure 4.2 shows the typical behavior of the noise injected into the model. It can be said with confidence that the noise generation process is performing as is required.

V. Conclusions and Recommendations

This chapter restates the major objectives of the thesis and indicates to what extent the objectives are met. The research effort is summarized and recommendations and future considerations are presented.

5.1 Conclusions

The primary goal of this research is to develop a high fidelity simulation of the LN-94 INS for use on a personal computer in the AFIT Navigation Lab. This goal is achieved. Some problems are encountered, however, and are outlined as the individual secondary goals are discussed.

The selection of a reduced order model of the LN-94 is one of the first tasks completed in the research. The intent of the reduced order model is reduction in computation loading. This is imperative in a simulation trying to achieve real-time processing on a PC class machine. The answer was found in research being conducted at AFIT. A 23-state LN-94 model was developed but was not implemented by Stacey [23,27]. Development of this model in this thesis is limited to implementation within the simulation and some tuning of noise processes to ensure a match with the actual INS operation.

Validation of the model using Britting's analysis is cursory at best, given the differences in implementation. Britting's data, however, is presented in such a way as to make the comparison with simulated data relatively easy. The characteristics of the 23-state

model are consistent with INS behavior, and simulation results bear out that conclusion. Differences between the LN-94 simulation and Britting's analysis are such that, for future research, the validation results provided in this research may be more appropriate.

One of the most critical goals of the thesis is the selection of an integration technique. This technique is responsible for the propagation of the model over time and model validity is directly impacted by its processing abilities. The fifth-order Kutta-Merson technique is chosen based on its ability to handle six-degree-of-freedom, trajectory-type integrations [7]. The automatic step sizing algorithm is an added advantage to the technique.

Problems were encountered with the development of a spline-fitting algorithm for the generation of trajectory estimates between data points. The varying dynamics of the fighter profile do not permit using fixed trajectory points and still achieve the necessary accuracy in the simulation. To obtain realistic results, a spline-fitting algorithm must be implemented. The programming of this algorithm is not achieved in this research. Errors in the coding could not be resolved. Synchronization of data with integration and the generation of corrupted data proved to be the greatest areas of concern and as a result, the fighter profile was not implemented.

INERCA [15] proves to be a very easy program to use but is limited in the output data set that it provides. This program should be expanded to include roll, pitch, and yaw data or be abandoned in favor of PROFGEN [2].

Stochastic noise generation is discussed in some detail in Chapter 4, Section 4.3. The system random number generator proves to be effective. Sequential correlation is essentially eliminated by using a random number to select from a vector of random numbers. If there is a weakness in this generator it is the size of RAND_MAX which in the case of PC's is 32768, where RAND_MAX represents the maximum positive integer type number achievable in the 16-bit environment. The consequence of this is that, for the large number of random numbers required, it is necessary to evaluate the same 32768 numbers n times, where n is the actual numbers required divided by 32768. Admittedly, the numbers are somewhat correlated by this process. To achieve better results, a different operating system is required.

One of the goals is to design an effective user interface. The user interface is achieved through a character-based, interactive question-and-answer approach. Sufficient information is provided by a program overview at the beginning of the program, and the questions are fairly self-explanatory. The interface allows selection of a trajectory type, type of states to track and store, and collection frequency. Plotting capabilities are accessible through the program interface and are provided by EZPLOT, an Air Force developed program. A user's guide is provided as Appendix F if any questions arise concerning the operation of the simulation.

The capability to communicate with the 1553B bus is achieved through the Ballard PC1553-2 interface card [5]. Formatted data according to the FNU 85-1 [3] is produced by the simulation. The

simulation responds to any one of three command words; #1, #2, and #24. The specifications for these command words are provided in Appendix A.

The final goal is the validation of the simulation. Validation results show that INS_LN94 C is a viable simulation of an actual LN-94 INS. Validation is performed for a static navigation trajectory and a straight-and-level trajectory. Both cases highlight the simulation's ability to emulate the LN-94. There are limitations to the validation process.

The static navigation case is compared against the data collected from an INS in an uncontrolled environment. This results in unknown influences on the INS which are not modelled in the simulation. Simulation noise processes can account for some of these unknown inputs but not all. Nonetheless, the simulation emulates the LN-94 within ±1% for position error, and behavior of all navigation parameters is similar.

The straight-and-level simulation tracks the true trajectory very well. Some might say too well. Without specific flight data to compare against, the accuracy of the simulation is based on the FNU 85-1 accuracy specifications. The simulation meets all the specifications but the question that arises is, "Did the simulation meet them too well or not well enough?" The flight data can show precisely how the LN-94 tracks a trajectory of this type. An INS of this caliber should follow the true trajectory very closely. A good flight data set would prove beneficial to the validation process.

5.2 Recommendations

The production of an INS simulation is the primary goal in this thesis and the degree of its fidelity is a measure of the success of the thesis. It is an excellent tool in which to exercise the knowledge acquired during the navigation sequence of the masters program. However, that is only one aspect of the research effort.

A good portion of this effort is in the software development of these principles into a cohesive and efficient simulation. This simulation development requires a substantial amount of programming: on the order of 5000 lines. It requires a broad knowledge in programming techniques in both higher level language and assembly language. A good working knowledge of the personal computer and the DOS operating system is required. The molding of the navigation principles into a software package with specific capabilities is a fascinating experience. Continued research in this area is highly recommended as simulation is the tool of the future.

It is also recommended that the software development that does go on as a result of research receive more attention in the overall evaluation of the research effort. Emphasis should be put on developing well managed, well documented, and efficient software development techniques.

The process of validating software is an on-going process. The validation of this software was sufficient to show that the simulation operates well as an emulation of the LN-94. However, given the magnitude problems in the model validation , and the somewhat uncontrolled environment of the static navigation validation, a more

precise evaluation tool is required to further substantiate the previous statement. It is strongly recommended that the validation process continue and an excellent tool for use in further validation is MSOFE [10].

The primary task should be to evaluate the 23-state model. By implementing the 23-state model in MSOFE on a VAX-based machine it is possible to compare results to ensure proper operation of the simulation software. This comparison also presents an opportunity for more tuning of the model and evaluation of initial conditions. It is important to continue evaluation of the initial conditions in an effort to reduce the phase differences that are seen between the simulated data and the actual INS data of the static navigation case.

The validation should first look at the error state behavior of the model to try and identify why the model is generating error magnitudes 10 times greater than the Britting analysis. When this situation has been clarified, MSOFE static navigation runs can be used to fine tune the simulation. This approach eliminates the unknown environment influences that are present when compared to actual INS data and provides the benchmark needed to tune the simulation in the static navigation case.

Another area where MSOFE may be used to advantage is through an implementation of the 23-state model in MSOFE on a PC-class machine. MSOFE can propagate the truth model in a similar fashion to the simulation, except with no real-time considerations. The results can provide a good indication of simulation capabilities. The MSOFE results

provide a benchmark in the PC environment, and therefore, an excellent tool for validation of the simulation software.

*5.2.1 **Future Research.*** The capabilities of a simulation are only limited by the creativity of the programmer. This simulation is no exception as the possibilities for future enhancements are limitless. It should form a solid basis for continued simulation research leading to an integrated navigation simulation package at the AFIT Navigation Laboratory.

In the short term, refinement of the program should be the next step in any research. The first step in refinement should come in the development of better trajectory data handling techniques. As this is the only area of difficulty in the development stage, this should receive attention first in any continued research. A close look at the synchronization issues of a spline-fitting algorithm would be beneficial. Synchronization difficulties and the generation of corrupted data are the main reasons this algorithm is not implemented. With the appropriate amount of time, this algorithm can be coded and its inclusion would enhance the simulation tremendously and allow for the use of highly dynamic trajectories.

Real-time simulation is another area which does not achieve full success. The static navigation case processes data in real time while the straight-and-level flight loses real time at a rate of approximately two seconds for every 30 minutes of navigation. Two approaches are possible to solve this issue. The first approach is to implement the simulation to take advantage of the 386 processing capabilities. This involves conversion of the source code to 386 compiler source code.

Recall the problems encountered with the WATCOM compiler, as discussed in Chapter 3. Care should be taken in selecting a compiler and depending on which compiler is used, this process may be simple or complex.

The second approach is to take advantage of the speed of assembly language. This approach tends to be more time consuming to implement and in most cases very difficult unless the programmer is very familiar with assembly language. It also leads to less maintainable or modifiable code in addition to being difficult to document. The first approach is recommended though both are viable solutions to real-time processing

Expansion of the communications capabilities of the simulation is another area of refinement. The simulation is presently restricted to responding to one input and two output messages from the 1553 bus. All words are not utilized in each case. Expansion of the data set to include all words in these messages and possibly respond to other messages is a challenge.

The simulation can benefit from an alignment simulation. The alignment could be developed to provide randomized initial conditions and the simulation could have the capability to start NAV mode in attitude ready or degraded NAV modes [3]. An alignment process could make the concept of degraded navigation meaningful and provide a better teaching tool. The randomized initial conditions could provide the capability of multiple different flights for analysis of data.

The integration of Kalman filtering into the simulation would further enhance its capability to emulate the LN-94. One concern,

though, is to not simply develop another MSOFE. The challenge of real-time propagation and filtering would be an excellent follow-on research effort.

The final goal of any further research would be to develop a truly integrated navigation system simulation in the AFIT Navigation Laboratory. The expansion and development of the limited GPS and CADC simulations now present in the laboratory with the INS simulation incorporated into one system would provide an exceptional research and learning tool.

Appendix A: FNU 85-1 Word Formats

This appendix provides a summary of the information that is necessary to format the data produced by the simulation for the MILSTD 1553. The command word specifications are supplied and the command word responses are outlined. The exact format of those words used by the simulation are also provided.

A.1 Command Word Specifications

Table A.1 Command Word Specifications

ID	ADDR	R/T	SUB-ADDR	CNT	Comments
Command Word #1	05h	T	00001	22	INU to CC
Command Word #2	05h	R	00010	4	CC to INU
Command Word #24	05h	T	11000	14	INU to CC

Note that the word counts are not the full 32 words that are displayed on the simulation screen. The screen will only update those words specified in the command word specification.

Table A.2 Response to Command Word #1

Word	Word Specification	Ref Output
1	INU Status Word	6
2	Time Tag Velocity (16,17,18)	99B
3	Time Tag Attitude (10,11,12,[illegible])	99A
4	Time Tag Present Position (6,7,8,9)	99C
5	Baro-Inertial Altitude	13
6	Present Position Latitude (MSP)	14
7	Present Position Latitude (LSP)	14
8	Present Position Longitude (MSP)	15
9	Present Position Longitude (LSP)	15
10	Pitch	7
11	Roll	8
12	True Heading	N/A
13	Platform Azimuth	104
14	Wander Angle	36
15	Magnetic Heading	N/A
16	North-South Velocity	10
17	East-West Velocity	11
18	Vertical Velocity	12
19	Ground Speed	N/A
20	North-South Acceleration	16
21	East-West Acceleration	17
22	Vertical Acceleration	18

Note that where N/A appears in the Output column this means that the word is not used in the simulation.

Table A.3 Response to Command Word #2

Word	Word Specification	Ref Input
1	Present Position Latitude (MSP)	60
2	Present Position Latitude (LSP)	60
3	Present Position Longitude (MSP)	61
4	Present Position Longitude (LSP)	61

Note that Inputs 60 and 61 are identical to Outputs 14 and 15. Refer to Table A.9 and Table A.10 for the formats.

Table A.4 Response to Command Word #24

Word	Word Specification	Ref Output
1	INU Status Word	6
2	Time Tag Body Rate (4,5,6)	99H
3	Magnetic Variation	N/A
4	Roll Rate	34
5	Pitch Rate	33
6	Yaw Rate	35
7	Roll Angle Acceleration	N/A
8	Pitch Angle Acceleration	N/A
9	Yaw Angle Acceleration	N/A
10	Longitudinal Acceleration	N/A
11	Lateral Acceleration	N/A
12	Normal Acceleration	N/A
13	Align Status	N/A
14	INU Vendor Ident	N/A

Note that an N/A in the output column means that the word is not applicable to the simulation.

A.2 Word Formats

Table A.5 Roll and Pitch Word Formats

		Output #7 Pitch
P	16	
	15	+0.0054931640625
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.00000000000
		Bit Value Degrees Positive Nose Up

		Output #8 Roll
P	16	
	15	+0.0054931640625
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.00000000000
		Bit Value Degrees Positive Right Wing Down

Table A.6 True Heading and N-S Velocity Word Formats

		Output #9 True Heading
P	16	
	15	+0.0054931640625
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.00000000000
		Bit Value Degrees Positive Clockwise from True North

		Output #10 N-S Velocity
P	16	
	15	+0.125
	14	+0.25
	13	+0.5
	12	+1
	11	+2
	10	+4
	9	+8
	8	+16
	7	+32
	6	+64
	5	+128
	4	+256
	3	+512
	2	+1024
	1	+2048
	0	-4096
		Bit Value ft/sec North Positive

Table A.7 E-W and Vertical Velocity Word Formats

		Output #11 E-W Velocity
P	16	
	15	+0.125
	14	+0.25
	13	+0.5
	12	+1
	11	+2
	10	+4
	9	+8
	8	+16
	7	+32
	6	+64
	5	+128
	4	+256
	3	+512
	2	+1024
	1	+2048
	0	-4096
		Bit Value ft/sec East Positive

		Output #12 Vertical Velocity
P	16	
	15	+0.0625
	14	+0.125
	13	+0.25
	12	+0.5
	11	+1
	10	+2
	9	+4
	8	+8
	7	+16
	6	+32
	5	+64
	4	+128
	3	+256
	2	+512
	1	+1024
	0	-2048
		Bit Value ft/sec Positive Up

Table A.8 Baro-Inertial Altitude and Time Tag Word Formats

		Output #13 Baro-Inertial Altitude
P	16	
	15	+2
	14	+4
	13	+8
	12	+16
	11	+32
	10	+64
	9	+128
	8	+256
	7	+512
	6	+1,024
	5	+2,048
	4	+4,096
	3	+8,192
	2	+16,384
	1	+32,768
	0	-65,536
		Bit Value ft Positive Up from Sea Level

		Output # 99 A,B,C,H Time Tag
P	16	
	15	50
	14	100
	13	200
	12	400
	11	800
	10	1600
	9	3200
	8	6400
	7	12800
	6	25600
	5	51200
	4	102400
	3	204800
	2	409600
	1	819200
	0	1638400
		Bit Value Microseconds

Note: Total value is offset by +32,768 ft

ie. +50,000 ft (actual) is represented by +17,232 ft (data)

Table A.9 Present Position Latitude Word Format

		Output #14 Most Significant Word
P	16	
	15	+0.0054931640625
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.00000000000
		Bit Value Degrees

		Output #14 Least Significant Word
P	16	
	15	+8.3819031715e-8
	14	+1.67638063430e-7
	13	+3.35276126861e-7
	12	+6.70552253723e-7
	11	+1.341104507446e-6
	10	+2.682209014892e-6
	9	+5.364418029785e-6
	8	+1.072883605957e-5
	7	+2.1457672119141e-5
	6	+4.2915344238281e-5
	5	+8.5830688476565e-5
	4	+1.71661376953125e-4
	3	+3.43322753906250e-4
	2	+6.86645507812500e-4
	1	+1.37329101562500e-3
	0	+2.7465820312500e-3
		Bit Value Degrees North Latitude Positive

Table A.10 Present Position Longitude Word Format

		Output #15 Most Significant Word
P	16	
	15	+0.005493164062 5
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.0000000000
		Bit Value Degrees

		Output #15 Least Significant Word
P	16	
	15	+8.3819031715e-8
	14	+1.67638063430e-7
	13	+3.35276126861e-7
	12	+6.70552253723e-7
	11	+1.341104507446e-6
	10	+2.682209014892e-6
	9	+5.364418029785e-6
	8	+1.072883605957e-5
	7	+2.1457672119141e-5
	6	+4.2915344238281e-5
	5	+8.5830688476565e-5
	4	+1.71661376953125e-4
	3	+3.43322753906250e-4
	2	+6.86645507812500e-4
	1	+1.37329101562500e-3
	0	+2.7465820312500e-3
		Bit Value Degrees North Latitude Positive

Table A.11 N-S and E-W Acceleration Word Formats

		Output #16 N-S Acceleration
P	16	
	15	+0.0078125
	14	+0.015625
	13	+0.03125
	12	+0.0625
	11	+0.125
	10	+0.25
	9	+0.5
	8	+1.0
	7	+2.0
	6	+4.0
	5	+8.0
	4	+16.0
	3	+32.0
	2	+64.0
	1	+128.0
	0	-256.0
		Bit Value ft/s^2 North Positive

		Output #17 E-W Acceleration
P	16	
	15	+0.0078125
	14	+0.015625
	13	+0.03125
	12	+0.0625
	11	+0.125
	10	+0.25
	9	+0.5
	8	+1.0
	7	+2.0
	6	+4.0
	5	+8.0
	4	+16.0
	3	+32.0
	2	+64.0
	1	+128.0
	0	-256.0
		Bit Value ft/s^2 East Positive

Note: When acceleration exceeds range of word, the output shall stay at maximum until acceleration returns to within range.

Note: 1g sensed is transmitted as zero.

Table A.12 Vertical Acceleration and Wander Angle Word Formats

		Output #18 Vertical Acceleration
P	16	
	15	+0.015625
	14	+0.03125
	13	+0.0625
	12	+0.125
	11	+0.25
	10	+0.5
	9	+1.0
	8	+2.0
	7	+4.0
	6	+8.0
	5	+16.0
	4	+32.0
	3	+64.0
	2	+128.0
	1	+256.0
	0	-512.0
		Bit Value ft/s^2 Positive Up

		Output #36 Wander Angle
P	16	
	15	+0.0054931640625
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.00000000000
		Bit Value Degrees Positive Counter-Clockwise True North to Y axis

Note: When acceleration exceeds range of word, the output shall stay at maximum until acceleration returns to within range.

Note: 1g sensed is transmitted as zero.

Table A.13 Platform Azimuth and INS Status Word Formats

		Output #104 Platform Azimuth
P	16	
	15	+0.0054931640625
	14	+0.010986328125
	13	+0.02197265625
	12	+0.04394531250
	11	+0.08789062500
	10	+0.17578125000
	9	+0.35156250000
	8	+0.70312500000
	7	+1.40625000000
	6	+2.81250000000
	5	+5.62500000000
	4	+11.2500000000
	3	+22.5000000000
	2	+45.0000000000
	1	+90.0000000000
	0	-180.00000000000
		Bit Value Degrees Positive Clockwise from Y axis

		Output #6 INS Status Word
P	16	
	15	Holding Brake Select
	14	Operational Mode
	13	Operational Mode
	12	Operational Mode
	11	Align Hold
	10	Taxi Hold
	9	Kalman Update Reject
	8	Kalman Update Complete
	7	Filter Background Busy
	6	Degraded Nav
	5	Bit Acknowledge
	4	Baro-Inertial Altitude Valid
	3	Time Tag Valid
	2	Battery Valid
	1	Attitude Valid
	0	INU Valid

Note: Op Mode (bit 12,13,14)--0 0 0 STANDBY, 0 0 1 NAV, 0 1 0 GC Align, 0 1 SH Align, 1 0 1 INFLIGHT Align

Appendix B: LN-94 Truth Model Definition

This appendix provides an overview of the 93-state error model defined in the Litton CDRL [17].

Table B.1 INS Truth Error Model Partition δx_1

State	Name	Definition
1	$\delta\theta_x$	x component of position error-true to computer frame
2	$\delta\theta_y$	y component of position error-true to computer frame
3	$\delta\theta_z$	z component of position error-true to computer frame
4	ϕ_x	x component of platform tilt error-true to platform frame
5	ϕ_y	y component of platform tilt error-true to platform frame
6	ϕ_z	z component of platform tilt error-true to platform frame
7	δV_x	x component of error in computed velocity
8	δV_y	y component of error in computed velocity
9	δV_z	z component of error in computed velocity
10	δh	error in vehicle altitude above reference ellipsoid
11	δh_L	error in lagged inertial altitude
12	δS_3	error in vertical channel
13	δS_4	error in vertical channel

Table B.2 INS Truth Error Model Partition δx_2

State	Name	Definition
14	b_{xc}	x component of gyro correlated drift rate
15	b_{yc}	y component of gyro correlated drift rate
16	b_{zc}	z component of gyro correlated drift rate
17	∇_{xc}	x component of accel and velocity quantizer correlated noise
18	∇_{yc}	y component of accelerometer and velocity quantizer correlated noise
19	∇_{zc}	z component of accelerometer and velocity quantizer correlated noise
20	δg_x	x component of gravity error
21	δg_y	y component of gravity error
22	δg_z	z component of gravity error
23	δh_c	barometer correlated bias noise error
24	b_{xt}	x component of gyro trend
25	b_{yt}	y component of gyro trend
26	b_{zc}	z component of gyro trend
27	∇_{xt}	x component of accel trend
28	∇_{yt}	y component of accel trend
29	∇_{zt}	z component of accel trend

Table B.3 INS Truth Error Model Partition δx_3

State	Name	Definition
30	b_x	x component of gyro drift rate repeatability
31	b_y	y component of gyro drift rate repeatability
32	b_z	z component of gyro drift rate repeatability
33	S_{gx}	x component of gyro scale factor error
34	S_{gy}	y component of gyro scale factor error
35	S_{gz}	z component of gyro scale factor error
36	χ_1	x gyro misalignment about y
37	χ_2	y gyro misalignment about x
38	χ_3	z gyro misalignment about x
39	υ_1	x gyro misalignment about z
40	υ_2	y gyro misalignment about z
41	υ_3	z gyro misalignment about y
42	D_{xxx}	x gyro scale factor non-linearity
43	D_{yyy}	y gyro scale factor non-linearity
44	D_{zzz}	z gyro scale factor non-linearity
45	S_{Qbx}	x gyro scale factor asymmetry error
46	S_{Qby}	y gyro scale factor asymmetry error
47	S_{Qbz}	z gyro scale factor asymmetry error

Table B.4 INS Truth Error Model Partition δx_4

State	Name	Definition
48	∇_{bx}	x component of accelerometer bias repeatability
49	∇_{by}	y component of accelerometer bias repeatability
50	∇_{bz}	z component of accelerometer bias repeatability
51 52 53	S_{Ax} S_{Ay} S_{Az}	x,y,z components of accelerometer and velocity quantizer scale factor error
54 55 56	S_{QAx} S_{QAy} S_{QAz}	x,y,z components of accelerometer and velocity quantizer scale factor asymmetry
57	f_{xx}	coefficient of error proportional to square of measured acceleration
58	f_{yy}	coefficient of error proportional to square of measured acceleration
59	f_{zz}	coefficient of error proportional to square of measured acceleration
60 61 62 63 64 65	f_{xy} f_{xz} f_{yx} f_{yz} f_{zx} f_{zy}	coefficients of error proportional to products of acceleration along and orthogonal to accelerometer sensitive axis
66	μ_1	x accel misalignment about z
67	μ_2	y accel misalignment about z
68	μ_3	z accel misalignment about x
69	σ_3	z accel misalignment about y

Table B.5 INS Truth Error Model Partition δx_5

State	Name	Definition
70 71 72	∇_{xq} ∇_{yq} ∇_{zq}	x,y,z components of accelerometer bias thermal transient
73 74 75	b_{xq} b_{yq} b_{zq}	x,y,z components of initial gyro drift rate bias thermal transient

Table B.6 INS Truth Error Model Partition δx_6

State	Name	Definition
76 77 78 79 80 81	F_{xyz} F_{xyy} F_{xyx} F_{xzy} F_{xzz} F_{xzx}	x gyro compliance terms
82 83 84 85 86 87	F_{yzx} F_{yzz} F_{yzy} F_{yxz} F_{yxx} F_{yxy}	y gyro compliance terms
88 89 90 91 92 93	F_{zxy} F_{zxx} F_{zxz} F_{zyx} F_{zyy} F_{zyz}	z gyro compliance terms

Appendix C: LN-94 Dynamic and Noise Matrices

This appendix provides a summary of the content of the dynamic matrix and the noise matrix of the 93-state LN-94 truth model.

	$\delta\theta_x$	$\delta\theta_y$	$\delta\theta_z$	ϕ_x	ϕ_y	ϕ_z	δV_x	δV_y	δV_z	δh	δh_L	δS_3	δS_4
$\dot{\delta\theta}_x$			$-\rho_y$					$-C_{RY}$					
$\dot{\delta\theta}_y$			ρ_x				C_{RX}						
$\dot{\delta\theta}_z$	ρ_y	$-\rho_x$											
$\dot{\phi}_x$		$-\Omega_z$	Ω_y		ω_z	$-\omega_y$		$-C_{RY}$					
$\dot{\phi}_y$	Ω_z		$-\Omega_x$	$-\omega_z$		ω_x	C_{RX}						
$\dot{\phi}_z$	$-\Omega_y$	Ω_x		ω_y	$-\omega_x$								
$\dot{\delta V}_x$	$-2V_y\Omega_y$ $-2V_z\Omega_z$	$2V_y\Omega_x$	$2V_z\Omega_x$		$-A_z$	A_y	$-V_zC_{RX}$	$2\Omega_z$	$-\rho_y-2\Omega_y$				
$\dot{\delta V}_y$	$2V_x\Omega_y$	$-2V_x\Omega_x$ $-2V_z\Omega_z$	$2V_z\Omega_y$	A_z		$-A_x$	$-2\Omega_z$	$-V_zC_{RY}$	$\rho_x+2\Omega_x$				
$\dot{\delta V}_z$	$2V_x\Omega_z$	$2V_y\Omega_z$	$-2V_y\Omega_y$ $-2V_x\Omega_x$	$-A_y$	A_x		$\rho_y+2\Omega_y$ $+V_xC_{RX}$	$-\rho_x-2\Omega_x$ $+V_yC_{RY}$		$\frac{2\ g_o}{a}$	$-k_2$	$-1.$	k_2
$\dot{\delta h}$									1.		$-k_1$		k_1-1
$\dot{\delta h}_L$										1.	$-1.$		
$\dot{\delta S}_3$											k_3		$-k_3$
$\dot{\delta S}_4$	$\dot{\delta S}_4 = (k_4-1)\ \delta S_4 + k_4\ \dot{\delta h}_L - k_4\ \dot{\delta h}_C \quad \dot{\delta h}_B =$ $\dot{\delta h}_c$, $\dot{\delta h}_B =$ δh_c												

Figure C.1 Elements of Dynamic Sub-matrix F_{11}

	$-b_{x_c}$	$-b_{y_c}$	$-b_{z_c}$	∇_{x_c}	∇_{y_c}	∇_{z_c}	$\delta\theta_x$	$\delta\theta_y$	$\delta\theta_z$	δh_c	$-b_{x_t}$	$-b_{y_t}$	$-b_{z_t}$	∇_{x_t}	∇_{y_t}	∇_{z_t}
$\dot{\delta\theta}_x$																
$\dot{\delta\theta}_y$																
$\dot{\delta\theta}_z$																
$\dot{\phi}_x$	C_{11}	C_{12}	C_{13}								C_{11}^t	C_{12}^t	C_{13}^t			
$\dot{\phi}_y$	C_{21}	C_{22}	C_{23}								C_{21}^t	C_{22}^t	C_{23}^t			
$\dot{\phi}_z$	C_{31}	C_{32}	C_{33}								C_{31}^t	C_{32}^t	C_{33}^t			
$\dot{\delta V}_x$				C_{11}	C_{12}	C_{13}	1.							C_{11}^t	C_{12}^t	C_{13}^t
$\dot{\delta V}_y$				C_{21}	C_{22}	C_{23}		1.						C_{21}^t	C_{22}^t	C_{23}^t
$\dot{\delta V}_z$				C_{31}	C_{32}	C_{33}			1.	k_2				C_{31}^t	C_{32}^t	C_{33}^t
$\dot{\delta h}$										k_1						
$\dot{\delta h}_L$																
$\dot{\delta S}_3$										$-k_3$						
$\dot{\delta S}_4$																

Figure C.2 Elements of Dynamics Sub-matrix F_{12}

	b_x	b_y	b_z	S_{θ_x}	S_{θ_y}	S_{θ_z}	x_1	x_2	x_3	v_1	v_2	v_3	D_{xxx}	D_{yyy}	D_{zzz}	S_{Obx}	S_{Oby}	S_{Obz}
$\delta\theta_x$																		
$\delta\theta_y$																		
$\delta\theta_z$																		
$\dot{\phi}_x$	C_{11}	C_{12}	C_{13}	$C_{11}\omega_x$	$C_{12}\omega_y$	$C_{13}\omega_z$	$C_{11}\omega_z$	$-C_{12}\omega_z$	$C_{13}\omega_y$	$C_{11}\omega_y$	$C_{12}\omega_x$	$C_{13}\omega_x$	$C_{11}\omega_x^2$	$C_{12}\omega_y^2$	$C_{13}\omega_z^2$	$\delta C_{11}\lvert\omega_x\rvert$	$\delta C_{12}\lvert\omega_y\rvert$	$\delta C_{13}\lvert\omega_z\rvert$
$\dot{\phi}_y$	C_{21}	C_{22}	C_{23}	$C_{21}\omega_x$	$C_{22}\omega_y$	$C_{23}\omega_z$	$C_{21}\omega_z$	$-C_{22}\omega_z$	$C_{23}\omega_y$	$C_{21}\omega_y$	$C_{22}\omega_x$	$C_{23}\omega_x$	$C_{21}\omega_x^2$	$C_{22}\omega_y^2$	$C_{23}\omega_z^2$	$\delta C_{21}\lvert\omega_x\rvert$	$\delta C_{22}\lvert\omega_y\rvert$	$\delta C_{23}\lvert\omega_z\rvert$
$\dot{\phi}_z$	C_{31}	C_{32}	C_{33}	$C_{31}\omega_x$	$C_{32}\omega_y$	$C_{33}\omega_z$	$C_{31}\omega_z$	$-C_{32}\omega_z$	$C_{33}\omega_y$	$C_{31}\omega_y$	$C_{32}\omega_x$	$C_{33}\omega_x$	$C_{31}\omega_x^2$	$C_{32}\omega_y^2$	$C_{33}\omega_z^2$	$\delta C_{31}\lvert\omega_x\rvert$	$\delta C_{32}\lvert\omega_y\rvert$	$\delta C_{33}\lvert\omega_z\rvert$
δV_x																		
δV_y																		
δV_z																		
δh																		
δh_L																		
δS_3																		
δS_4																		

Figure C.3 Elements of Dynamics Sub-matrix F_{13}

Figure C.4 Elements of Dynamics Sub-matrix F_{14}

	∇_{x_q}	∇_{y_q}	∇_{z_q}	b_{x_q}	b_{y_q}	b_{z_q}
$\dot{\delta\theta}_x$						
$\dot{\delta\theta}_y$						
$\dot{\delta\theta}_z$						
$\dot{\phi}_x$				C_{11}	C_{12}	C_{13}
$\dot{\phi}_y$				C_{21}	C_{22}	C_{23}
$\dot{\phi}_z$				C_{31}	C_{32}	C_{33}
$\dot{\delta V}_x$	C_{11}	C_{12}	C_{13}			
$\dot{\delta V}_y$	C_{21}	C_{22}	C_{23}			
$\dot{\delta V}_z$	C_{31}	C_{32}	C_{33}			
$\dot{\delta h}$						
$\dot{\delta h}_L$						
$\dot{\delta S}_3$						
$\dot{\delta S}_4$	(NON-LINEAR EQUATION)					

Figure C.5 Elements of Dynamics Sub-matrix F_{15}

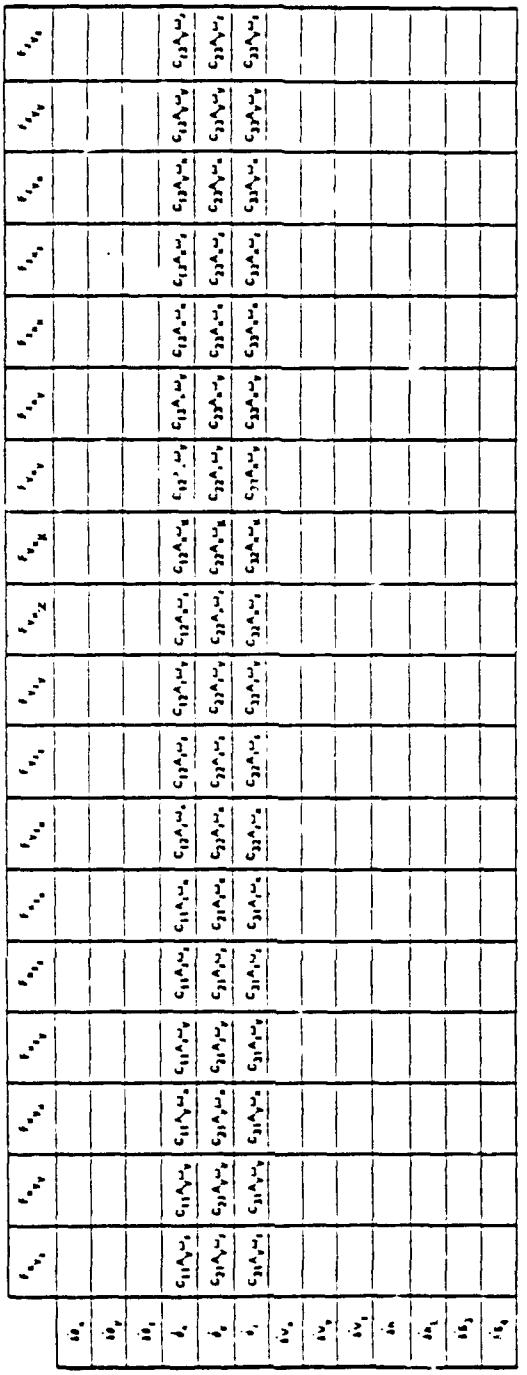


Figure C.6 Elements of Dynamics Sub-matrix F_{16}

	b_{x_c}	b_{y_c}	b_{z_c}	∇_{x_c}	∇_{y_c}	∇_{z_c}	δg_x	δg_y	δg_z	δh_c
$\dot{b}_{x_c}$	$-\beta b_{x_c}$									
$\dot{b}_{y_c}$		$-\beta b_{y_c}$								
$\dot{b}_{z_c}$			$-\beta b_{z_c}$							
$\dot{\nabla}_{x_c}$				$-\beta \nabla_{x_c}$						
$\dot{\nabla}_{y_c}$					$-\beta \nabla_{y_c}$					
$\dot{\nabla}_{z_c}$						$-\beta \nabla_{z_c}$				
$\dot{\delta g}_x$							$-\beta \delta g_x$			
$\dot{\delta g}_y$								$-\beta \delta g_y$		
$\dot{\delta g}_z$									$-\beta \delta g_z$	
$\dot{\delta h}_c$										$\beta \delta h_c$

Figure C.7 Elements of Dynamics Sub-matrix F_{22}

	∇_{x_q}	∇_{y_q}	∇_{z_q}	b_{x_q}	b_{y_q}	b_{z_q}
$\dot{\nabla}_{x_q}$	$-\beta\nabla_{x_q}$					
$\dot{\nabla}_{y_q}$		$-\beta\nabla_{y_q}$				
$\dot{\nabla}_{z_q}$			$-\beta\nabla_{z_q}$			
$\dot{b}_{x_q}$				$-\beta b_{x_{q_1}}$		
$\dot{b}_{y_q}$					$-\beta b_{y_{q_1}}$	
$\dot{b}_{z_q}$						$-\beta b_{z_{q_1}}$

Figure C.8 Elements of Dynamics Sub-matrix F_{55}

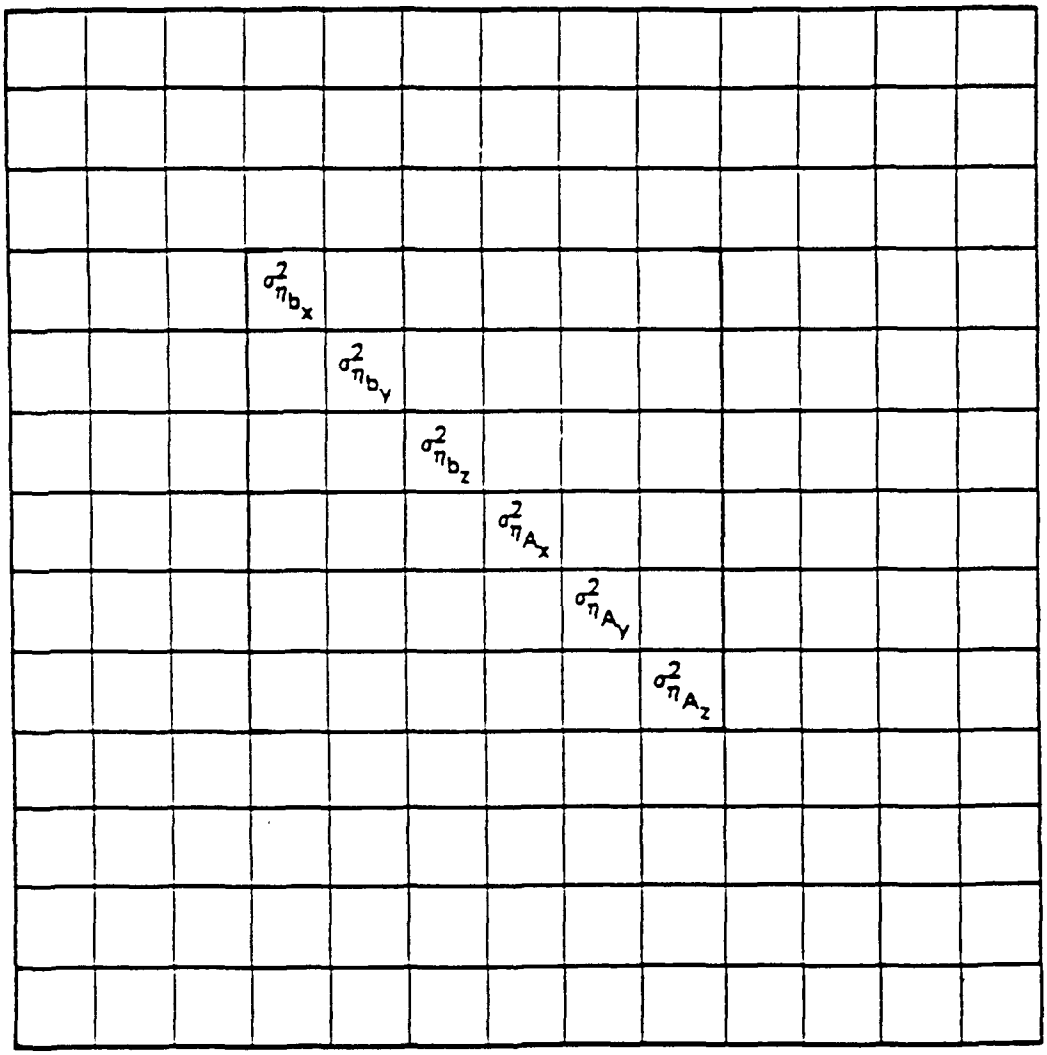
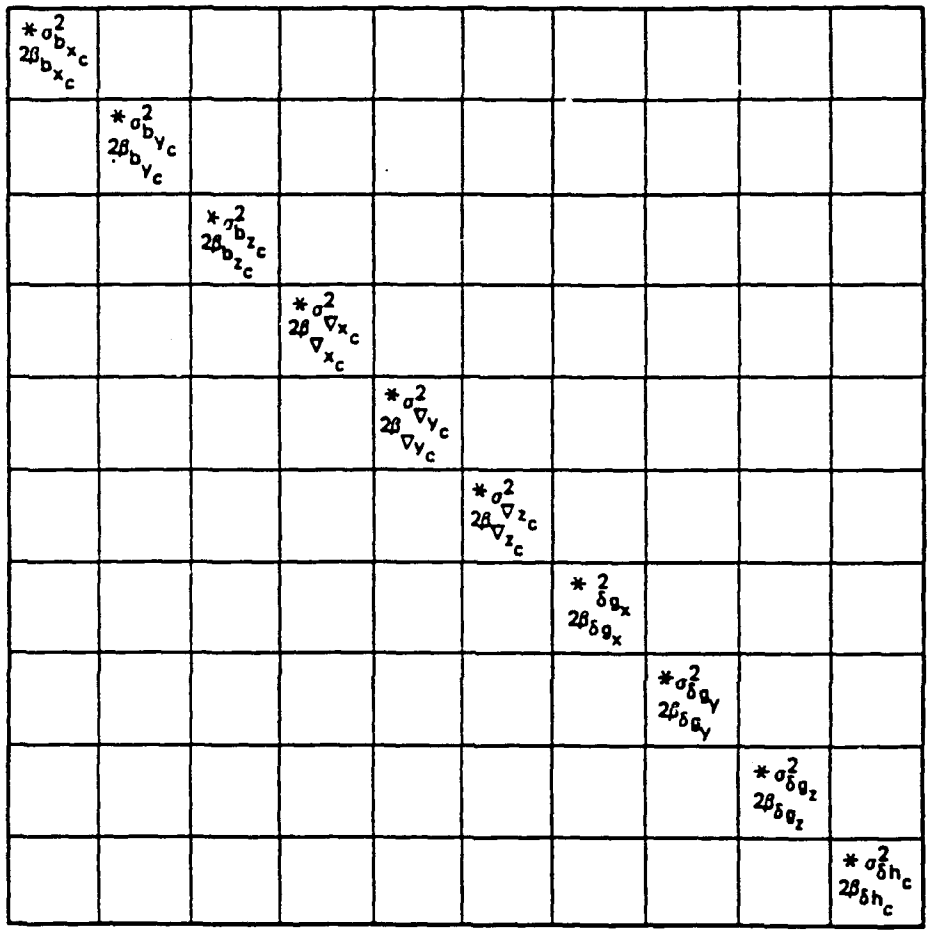


Figure C.9 Elements of Process Noise Matrix Q_{11}



* DENOTES PRODUCT eg: $2\beta_{b_{x_c}}\sigma^2_{b_{x_c}} = 2\beta_{b_{x_c}}\sigma^2_{b_{x_c}}$

Figure C.10 Elements of Process Noise Matrix Q_{22}

Appendix D: Programming Flow Charts

This appendix supplies the reader with program flow charts. These charts outline the program flow used in the more critical/complex functions in the simulation. For specific coding techniques the reader should refer to the source code.

For all functions except MAIN, there are two functions outlined in the upper left corner of the first page of a flow chart. The function to the far left is the calling function and the next function is the one being called and displayed as a flow chart. For the function MAIN, only the function MAIN is shown at the upper left of the first page of the flow chart.

D.1 Function MAIN

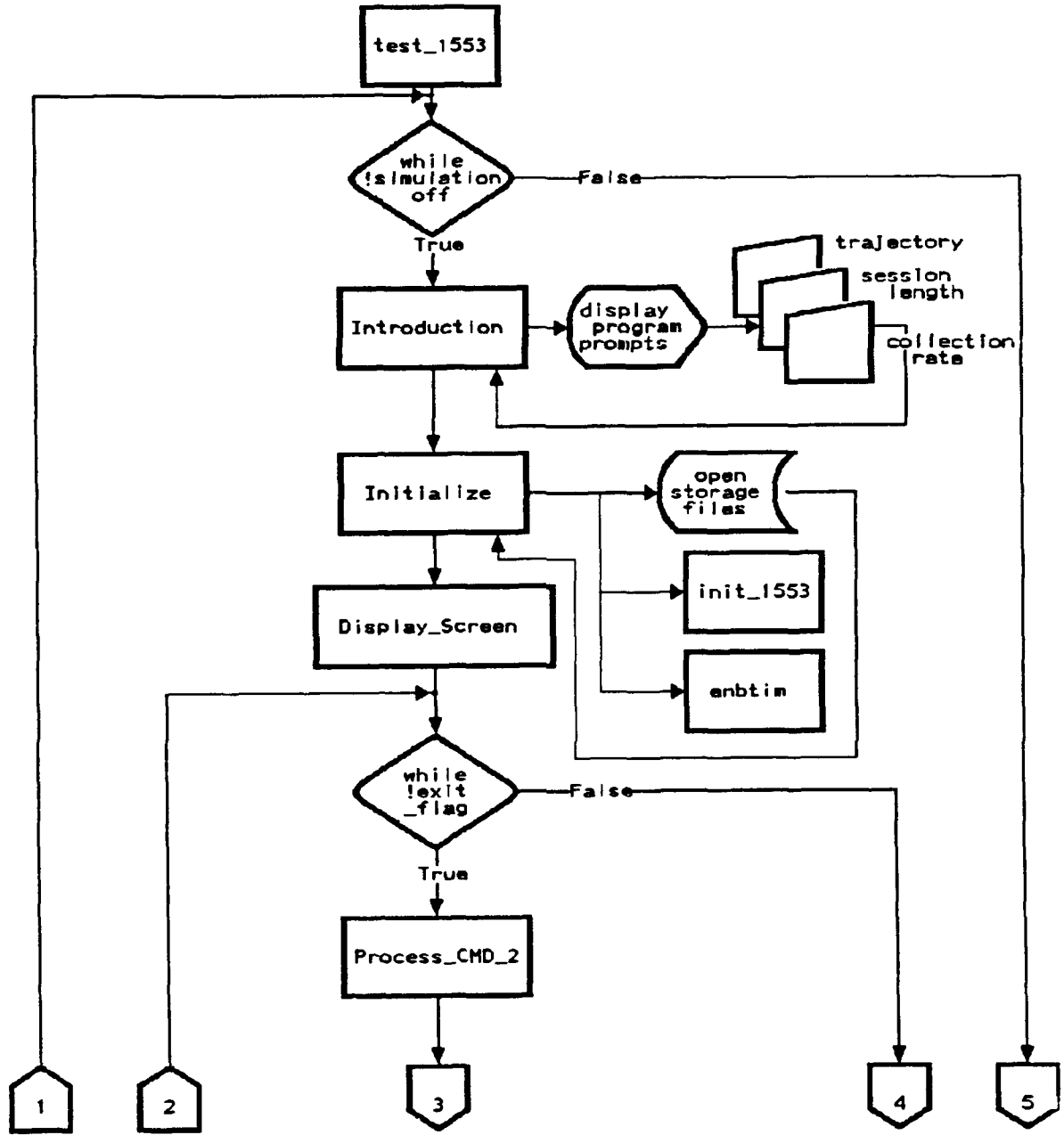


Figure D.1 Function MAIN Part 1

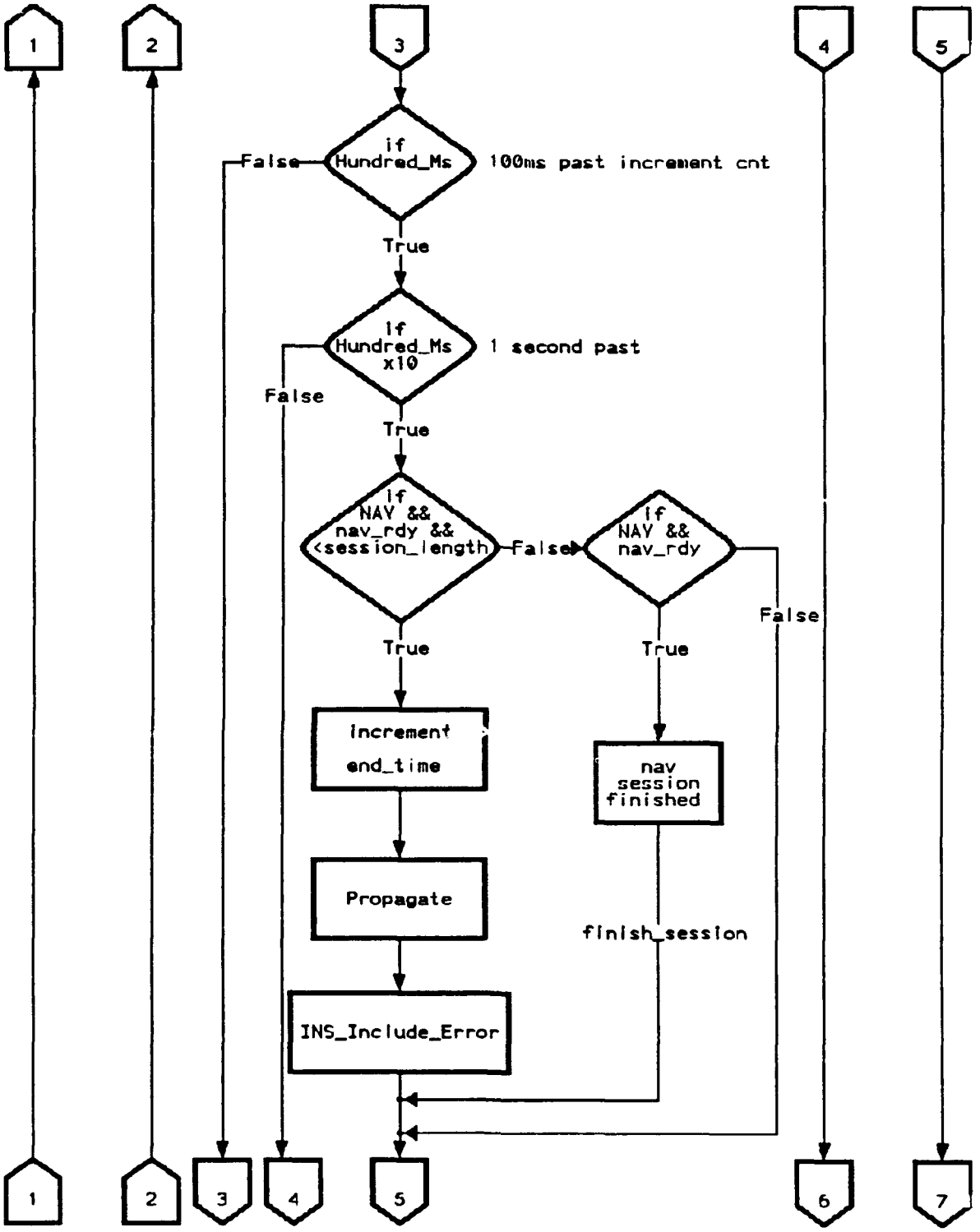


Figure D.2 Function MAIN Part 2

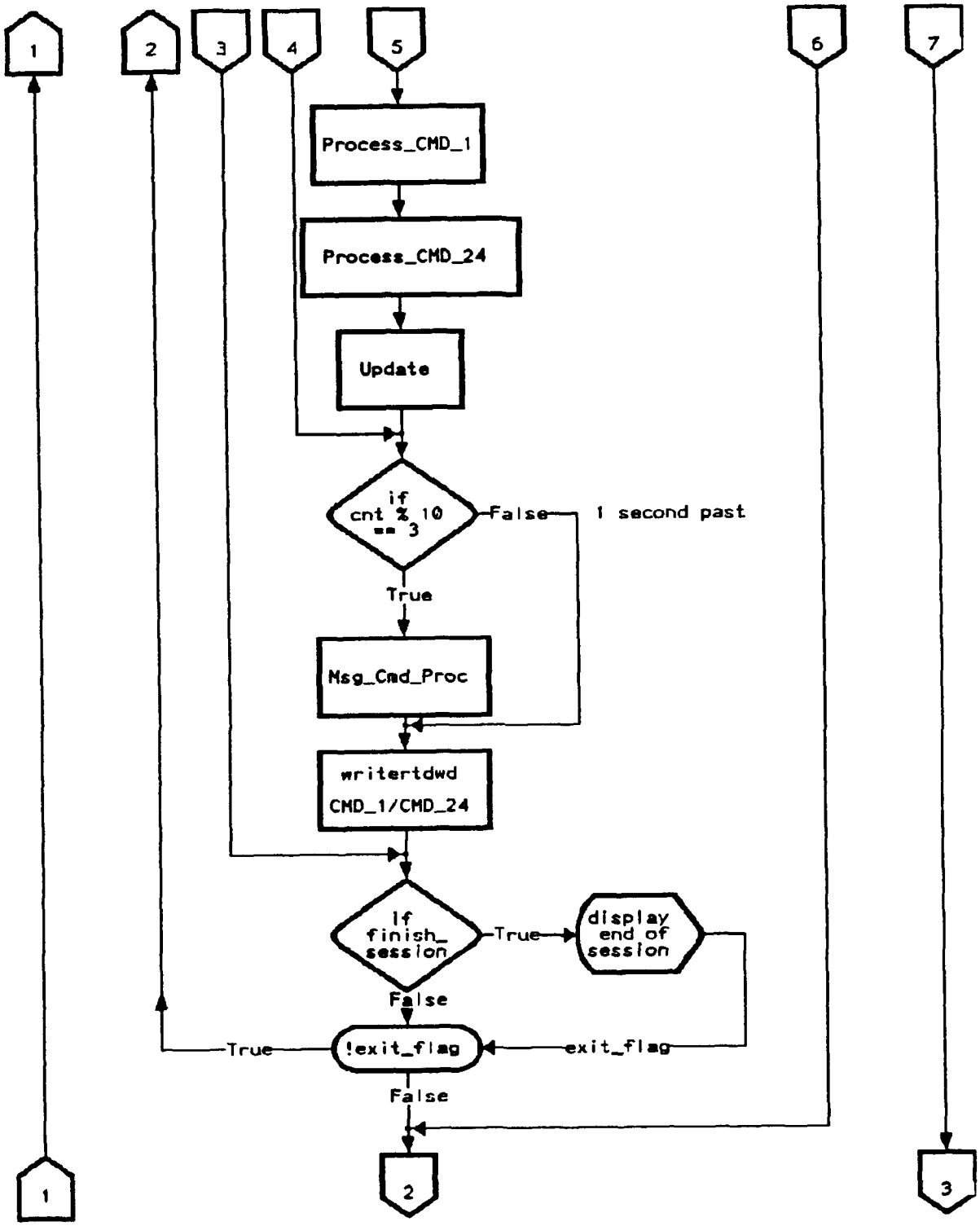


Figure D.3 Function MAIN PART 3

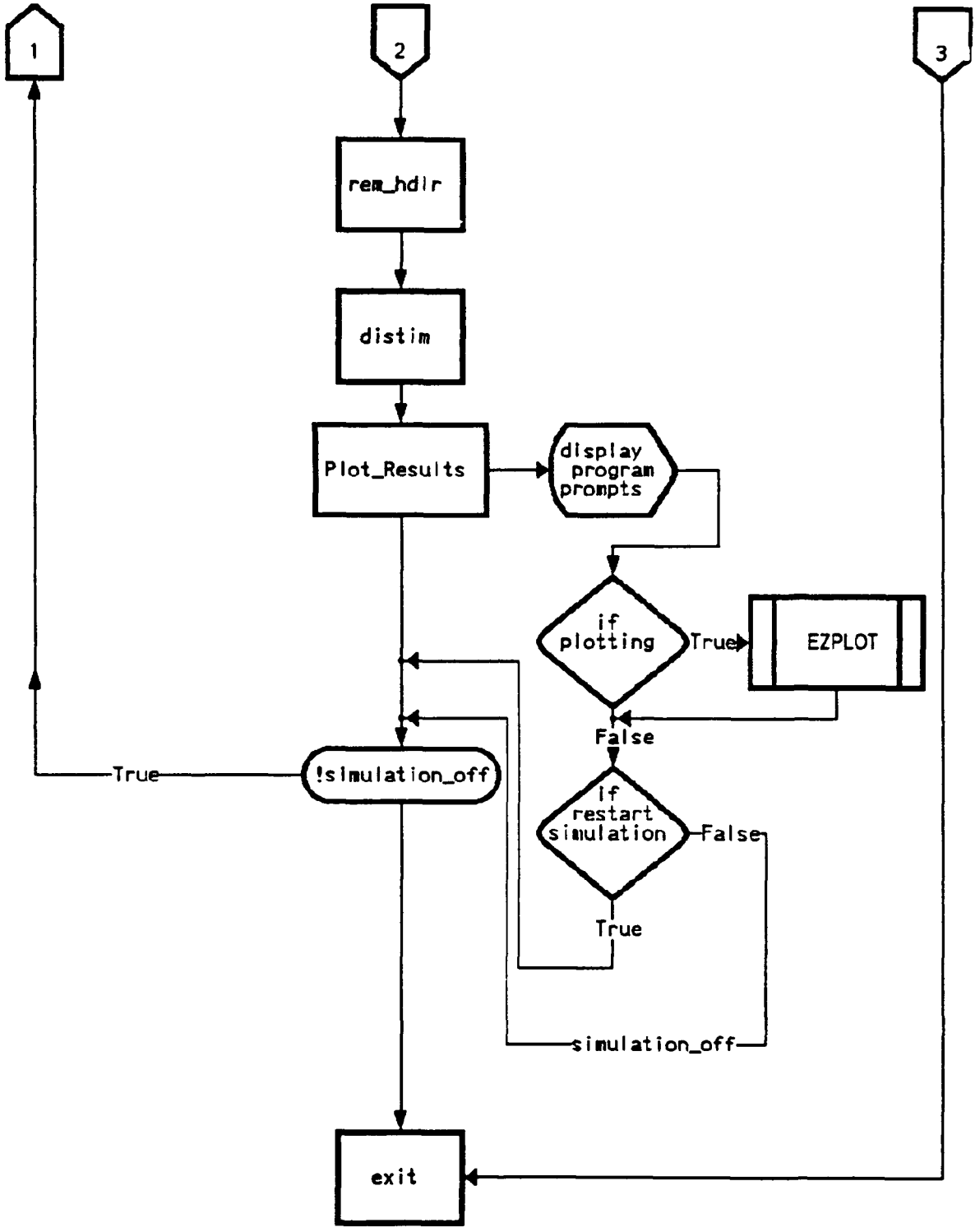


Figure D.4 Function MAIN Part 4

D.2 Function PROPAGATE

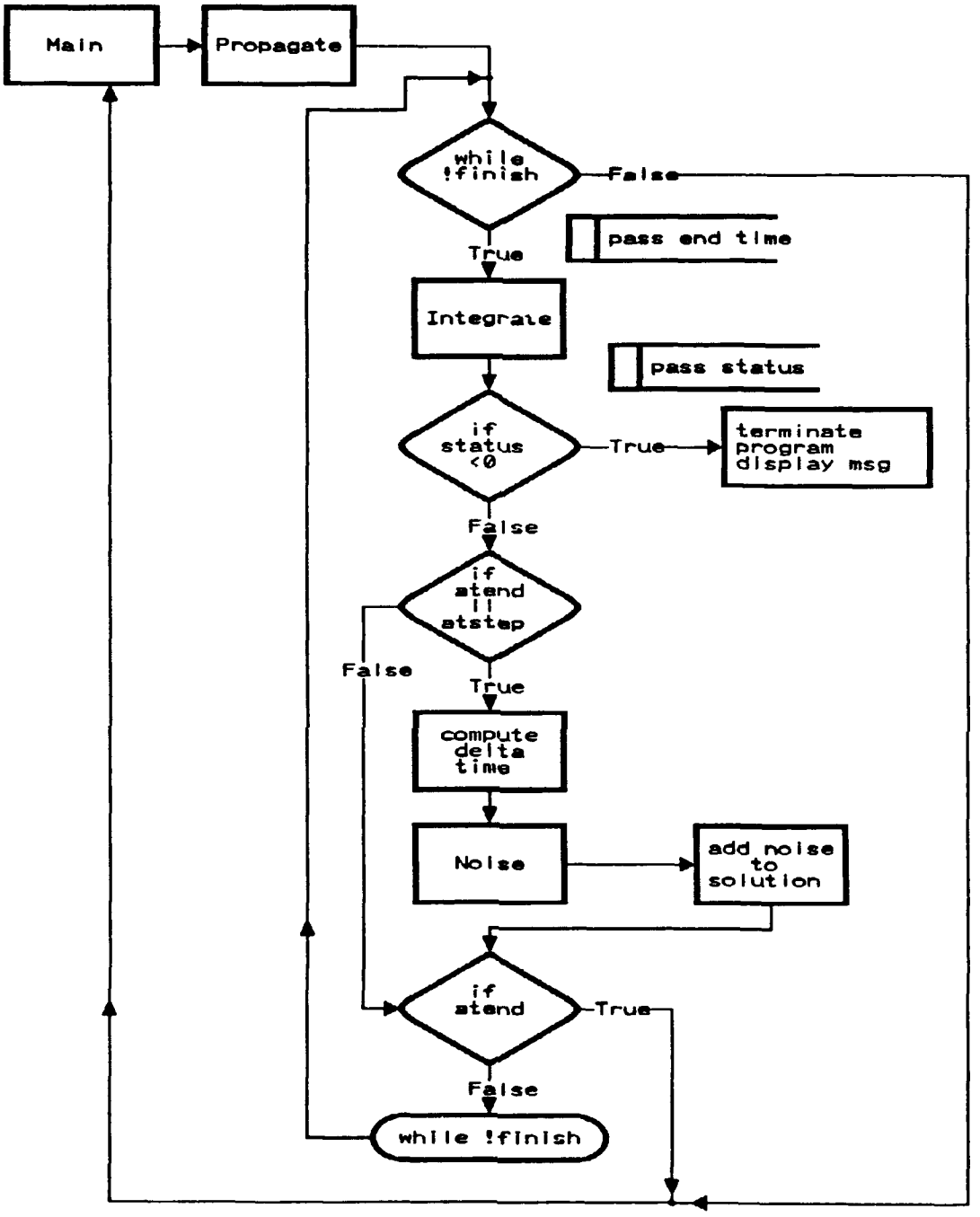


Figure D.5 Function PROPAGATE

D.3 Function INTEGRATE

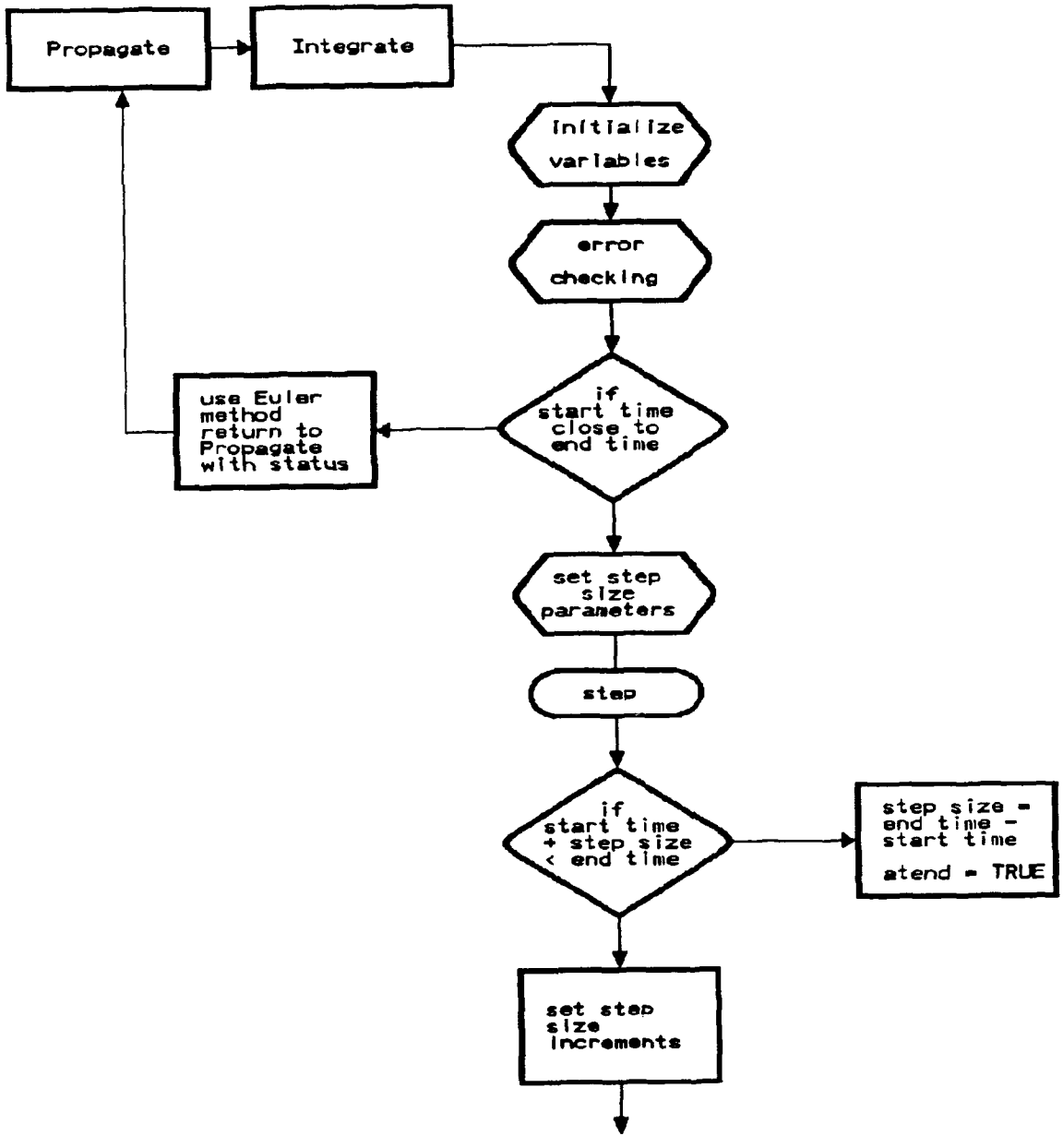


Figure D.6 Function INTEGRATE Part 1

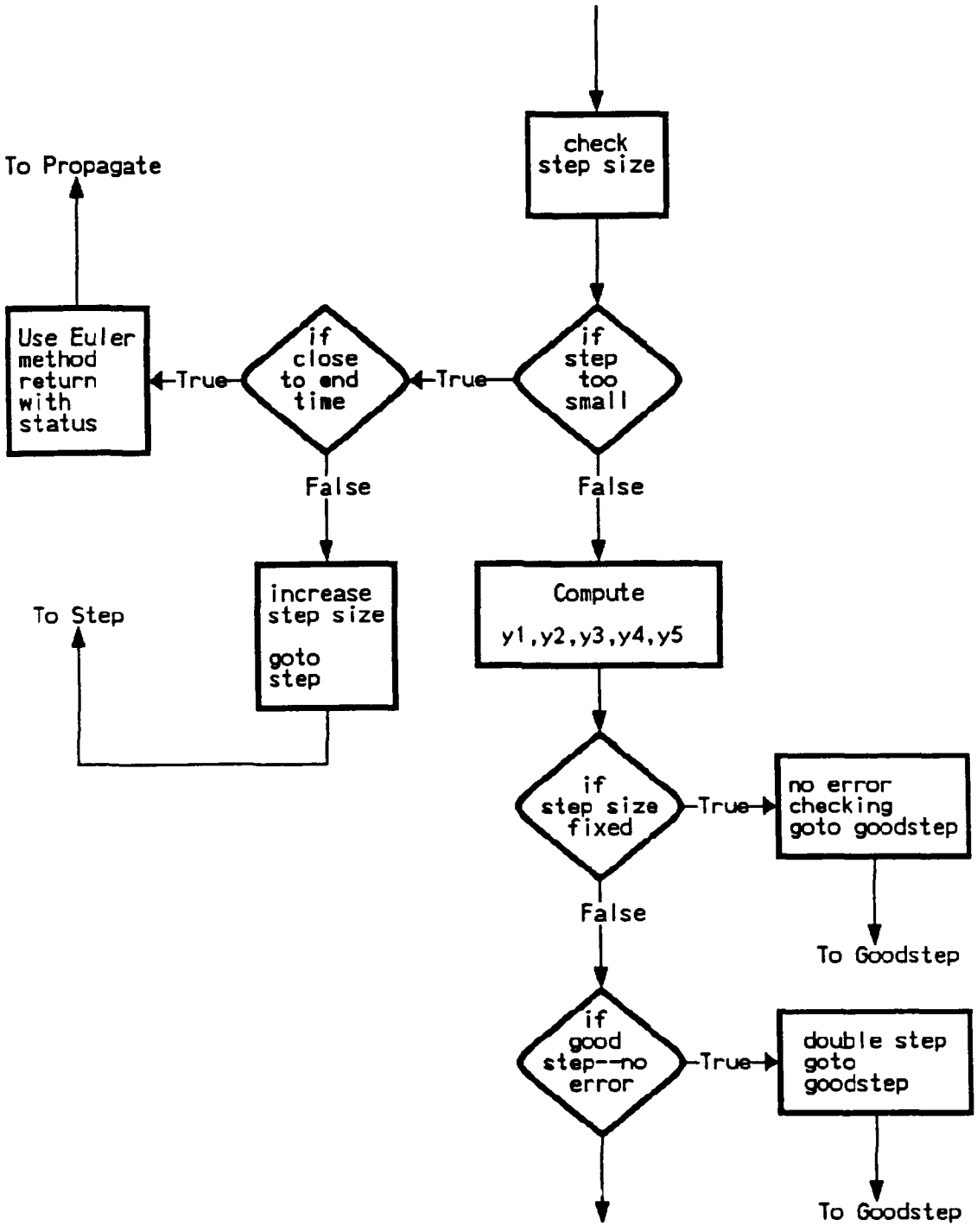


Figure D.7 Function INTEGRATE Part 2

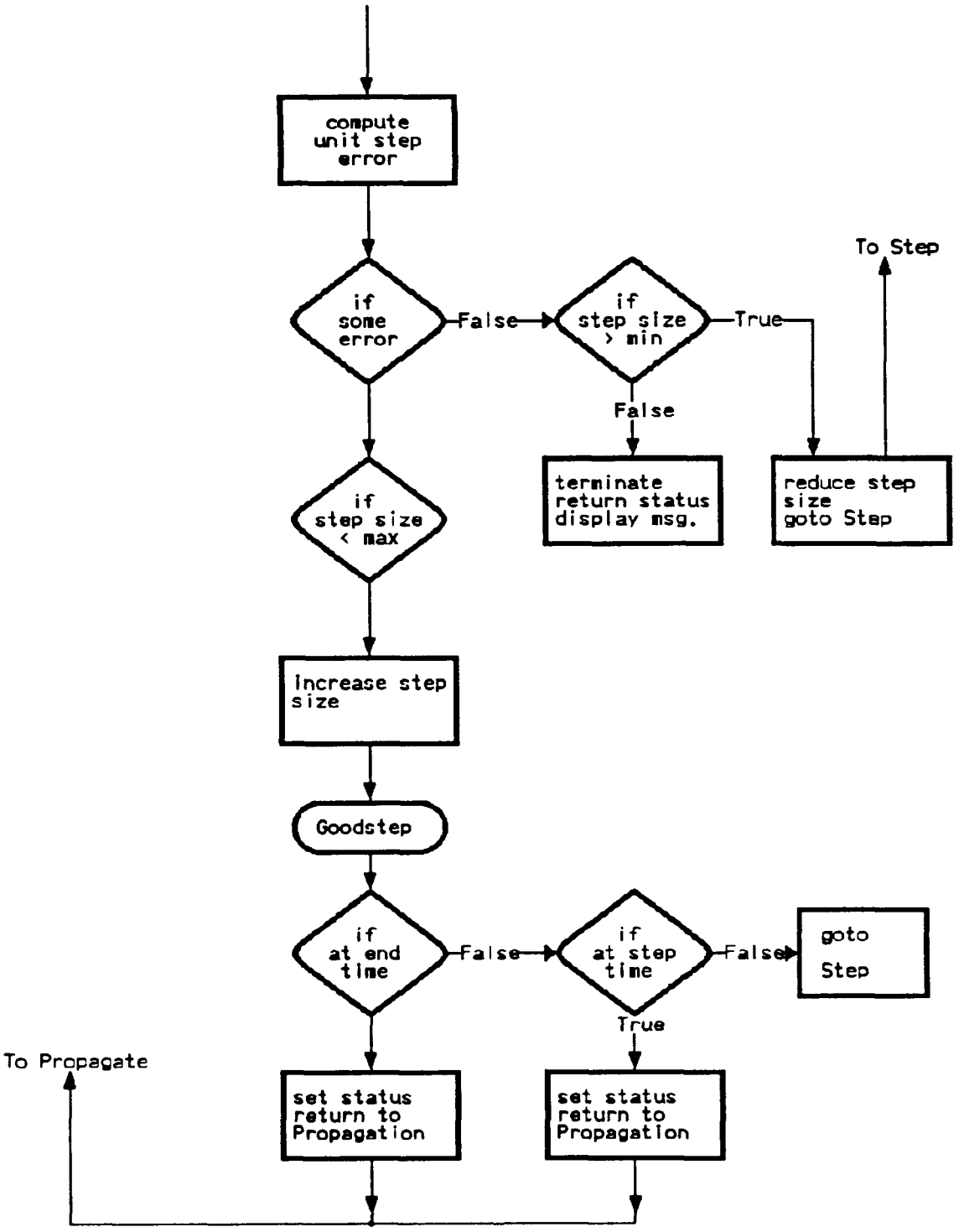


Figure D.8 Function INTEGRATE Part 3

Appendix E: Program Functions

Chapter 3 provides the global overview of the simulation program. In this appendix, the program is broken down into functions as they relate to the above mentioned modules and how they relate to other program functions. Refer to Table E.1 and Table E.2 for a breakdown of the relationship between the functions and the modules. The intent is not to provide code descriptions, but rather to describe the general operation of the function. The reader is referred to the program source code for code level descriptions.

The INS_LN94 executable program consists of the INS_LN94 source code which contains the MAIN function and is linked with five object modules:

1. SETUP.OBJ
2. PROCESS.OBJ
3. MCDRVL.OBJ
4. INS_INT2.OBJ
5. CLKT10.OBJ

There are 35 functions written specifically for the simulation not including the MAIN. In addition the program uses 10 commercially developed functions. Specific discussion of the commercially developed functions is not provided other than to indicate where they are used and then reference the applicable documentation.

For simplicity, the functions are presented in alphabetical order. The function is identified by name followed by the name of the host program and any calling function names. A description of function operation then follows. First, a look at functions written specifically

Table E.1 Functions Contained in INS_LN94.C

	INS_LN94.C	
Bitoff	Biton	Calc_Time_Tag
Derivative	Display_Lat_Long	Extract
Hundred_Ms	Initialize	INS_Error_Include
Insert	Integrate	Keyboard
Matrix_Ops	Mode_Change_Timer	Model
Msg_Equal	Msg_Comd_Proc	Noise
Propagate	Random_Number	Trajectory
Update	Valid_Msg_Number	Write_Data

Table E.2 Functions Contained in Object Modules

SETUP.C	PROCESS.C	CLKT10.ASM	INS_INT2.ASM
Introduction	Process_CMD_1	enbtim	init1553
Disp ..y_Screen	Process_CMD_2	distim	rem_hdlr
Plot_Results	Process_CMD_24		
test_1553			

for the simulation.

E.1 Program Specific Functions

E.1.1 Bitoff

Host Program: INS_LN94.C
Calling Function(s): Process_CMD_1

Calling Convention: Bitoff(unsigned int,int)

Function Bitoff is used to set specified bits in a word to zero. A word of unsigned int is passed together with a bit mask value to the

function. The word value is AND'd with the compliment of the mask value, and the result is returned with the designated bit equal to zero. The mask values are defined in INSDEF.H and are simply hex values which highlight the bit in question. For example the masked value of bit one would be 0x8000, where bit one is the most significant bit.

E.1.2 Biton

Host Program: INS_LN94.C
Calling Function(s): Process_CMD_1

Calling Convention: Biton(unsigned int,int)

Biton works in a similar fashion to Bitoff except the object is to set the masked bit to a value of one. This is achieved by using a bitwise exclusive OR of the word value and the mask.

E.1.3 Calc_Time Tag

Host Program: INS_LN94.C
Calling Function(s): Process_CMD_1

Calling Convention: Calc_Time_Tag(unsigned int)

The time tag is defined to be the difference between the process start time, and the time the sync mode command is received from the bus. The sync word is passed when the bus interface card is initialized in the function Initialize. The units of the time tag are in microseconds, with the least significant bit or truncation point at 50 microseconds. In order for the time tag to fit into a 16 bit entity, the calculations of time must be limited. In other words, the intermediate values of time must be limited to 419 in order to achieve a value that will fit into the word size when called for.

The difference between process time and sync time is calculated and from this value, microsecond units are derived, and the value

truncated to the nearest 50 microsecond. Time is then returned to the calling function.

E.1.4 Derivative

Host program: INS_LN94.C
Calling function(s): Integrate

Calling Convention: Derivative(double[])

Derivative is called by Integrate to evaluate the system derivative at time t. The system derivative is defined as the dynamic matrix F post multiplied by the state vector. To evaluate the derivative, the dynamic matrix and the state vector must be updated to time t.

To achieve this, the state vector is set to the previous integration solution, and then the function Trajectory is called to update the navigation parameters. Using the updated parameters Model is called to evaluate the system dynamic matrix at time t. Derivative then calls Matrix_Ops to perform the matrix multiplication. The new system derivative is then passed back to Integrate to continue propagation of the model.

Note that the stochastic white noise, which forms a part of the model differential equation, is dealt with in another function and noise is injected as per the initial conditions.

E.1.5 Display_Lat_Long

Host program: INS_LN94.C
Calling function(s): Process_CMD_1
Process_CMD_2
Update

Calling Convention:

Display_Lat_Long(unsigned int,unsigned int,int,int,int)

Display_Lat_Long is called to display the present position lat and long on the simulation screen. The most significant and least significant words of either lat or long are passed to the function along with the screen position in row/column format. Additionally, the type LAT or LONG is passed to the function.

Using a shift operation, the MSP and LSP are combined into one word. The value is checked for positive or negative value and if negative, a 2's compliment operation is performed and the sign value set. The value is then scaled according to FNU 85-1 and the degrees, minutes, and 1000th of minutes extracted. The direction and value are then displayed to the screen based on the type that has been passed to the function. Note that by convention, N latitude and E longitude are positive.

E.1.6 Display_Screen Host program: SETUP.C
Calling function(s): Main

Calling Convention: Display_Screen()

This function makes use of the Essential Software C Utility Library functions to manipulate screen color, cursor position, and formatted output to set the computer terminal screen for use of the simulation. The function also calls Write_Data to display the initial data in the input message.

This function is called by Main at the beginning of the program to set the screen and then all other screen manipulations are done by other functions as required.

E.1.7 distim Host program: INS_INT2.ASM
Calling function(s): Main

Calling Convention: distim()

This function is written in assembly language and is used at the end of the program to disable the simulation timer and reset the interrupt handler to operate at 18.2 Hz.

E.1.8 enbtim

Host program: INS_INT2.ASM
Calling function(s): Main

Calling Convention: enbtim(int far *)

This function sets up a 100 Hz clock for timing in the simulation. The handler for DOS interrupt 0 is replaced and the count of timer 0 is decreased to the value such that the timer interrupt occurs every 10 msec. The original interrupt handler is called at 18.2 Hz.

This function also stores a pointer to a count value, which the interrupt handler will increment every time it is called. This pointer is monitored to calculate the simulation time.

E.1.9 Extract

Host program: INS_LN94.C
Calling function(s): Insert

Calling Convention: Extract(unsigned int,int,int)

A word, a bit start, and an end location are passed to the function. Taking advantage of bitwise left and right shifts, the function Extract removes bits from a specified word starting at the designated start bit to the designated end bit. All undesignated bits are shifted out of the word and the remaining value is passed back to the calling routine.

E.1.10 Hundreds_Ms

Host Program: INS_LN94.C
Calling Function(s): Main

Calling Convention: Hundred_Ms()

This function is the basis for the timing in the simulation. The timer is set to interrupt at 10 ms intervals. Hundred_Ms tracks the 10 ms count and when 100 ms has past, the function will return TRUE. Otherwise the function will return FALSE. This type of timing sequence, in essence, means that the Main function can be operated based on increments of 100 ms.

Since the maximum 10 ms count for a 16 bit unsigned integer is 65535 the maximum simulation time would be slightly greater than 10 minutes. To alleviate this restriction Hundred_Ms tracks a rollover count. When the 65535 limit is reached a count of 65535 is added to the time for computation. This technique is used when tracking the difference between real time and processing time. This is a debugging technique and is used in the function Update. For program development this time is shown on the screen.

E.1.11 init1553 Host Program: CLKT10.ASM
Calling Function(s): Initialize

Calling Convention: init1553(int far *, int far *,int far *,int far *,int)

This function initializes the Ballard PC1553 interface board to RT mode and sets the RT address. An interrupt handler specifically used for the Ballard board is also installed. When a synchronize mode code is received, the handler will set a variable to current time which in turn synchronizes the board with the bus. In this function, this variable is initially set to zero.

E.1.12 Initialize Host Program: INS_LN94.C
Calling Function(s): Main

Calling Convention: Initialize()

The function, as the name implies, is called by Main to initialize program variables. It also initiates the simulation timer using enbtim and it initializes the PC1553 interface board using init1553. The function sets the initial conditions of the state vector and sets the operating parameters for the function Integrate. The function also initializes the 1553 message values to zero and opens the appropriate files for storage of data as requested in the program setup.

E.1.13 INS_Error_Include Host Program: INS_LN94.C
Calling Function(s): Main

Calling Convention: INS_Error_Include()

INS_Error_Include correlates true trajectory data with the INS errors that are propagated in the simulation. By convention INS-indicated information is the difference between true trajectory and INS errors. This function performs the operations necessary to produce INS-indicated values. It also makes the appropriate transformations to provide information in the proper reference frame. It is also the point in the program where the INS-indicated values or the error state values are stored to file.

Specifically, the function calculates the errors in latitude, longitude, wander angle, velocity, acceleration, pitch, roll, and platform azimuth, and pitch, roll, and yaw rates. These calculation are made from the solution vector of Integrate and transformed into the proper reference frame. The difference calculation is made between the true trajectory and INS error and as designated by the user, either the error states or the whole states are stored in files.

E.1.14 Insert

Host Program: INS_LN94.C
Calling Function(s): Keyboard

Calling Convention: Insert(int,unsigned int,int,int)

Function Insert pushes a designated set of bits or value into an integer word. These two parameters, as well as the start and end position within the integer word, are passed to the function. This function utilizes Extract to remove start-1 bits from the word and places them in a temporary word. The value or designated bits are then placed in the temporary word and Extract is used to place the end bit+1 bits of the word into the temporary word. The new word is passed back to the calling function.

E.1.15 Integrate

Host Program: INS_LN94.C
Calling Function(s): Propagate

Calling Convention: Integrate(double)

Integrate is the most complex function in the program and is the function responsible for propagating the system model and providing the resulting INS errors. Integrate finds the solution to the system differential equation from the present time to the end time which is passed by the calling function. It uses a fifth-order Kutta-Merson routine with automatic step sizing to achieve this solution.

Integrate first checks to ensure variables declared in Initialize are within the function limits. If this is not the case the function will terminate and the simulation will terminate. The parameters that are checked are start time versus end time; time cannot propagate backwards. The maximum step size is checked and must be greater than minimum step size, and the control flag can only be zero or one. An

additional parameter that is provided but not checked is the error tolerance used in the error checking phase of the program. The control flag that is indicated refers to check_step which when set will control step size to be either automatic step control or fixed step.

If integration time (same as present time) is very close to end time the function uses the Euler method of integration to obtain the solution. Otherwise, the function increases the step size and uses the Kutta-Merson equations to find the solution. If integrate time is not too small the routine moves directly to the Kutta-Merson equations. The Kutta-Merson equations, as outlined in Chapter 2, involve calling Derivative to evaluate the system derivative for time t. Time is segmented into step fractions of 1/2, 1/3, 1/6, and 1/8 in addition to time zero.

Once the solution has been obtained, Integrate uses an error checking routine to determine whether the solution is viable. The function evaluates the difference between y_4 and y_5 and computes an error estimate. If no error is calculated, the function will increase step size for the next iteration. If a successful step with some error, the function evaluates whether an increase in step size is viable without violating error tolerances. If an unsuccessful step the function will reduce the step size and try re-evaluating. If under the present parameters a solution can not be achieved, the function will terminate and return a value of the component which can not be evaluated within parameters.

After a successful step and appropriate step size, the function evaluates whether at the end time or at step time. If initial

conditions dictate noise at each step the function will return to Propagate to inject noise into the solution. If not at step and not at end time the function will continue integration.

In all cases when the function terminates, the status of the integration is returned for error handling. Refer to Section 3.3.1 for more detail on this function.

E.1.16 Introduction

Host Program: SETUP.C
Calling Function(s): Main

Calling Convention: Introduction()

This function interrogates the user at the commencement of the program asking the user to define a number of parameters. It opens with an overview of the simulation and its operation. This overview may be omitted at the discretion of the user. The user is asked to define the type of trajectory input for the flight, the type of states to track and store, the length of the flight, the rate of data collection, and then whether to carry on with the simulation as defined. The user is given the opportunity to make changes to the above parameters before commencing or exiting the program if desired.

The function uses formatted input and output to control the question and answer format and includes fail-safe techniques if the user inputs unexpected information.

E.1.17 Keyboard

Host Program: INS_LN94.C
Calling Function(s): Msg_Cmd_Proc

Calling Convention: Keyboard(int)

This function is called by Msg_Cmd_Proc to accept and interpret keyboard inputs. It accepts all valid inputs as defined by the

simulation. Specifically it accepts the ESC key, D01, I01, I24, F1, F2, F3, and F4. In the case of the function keys Keyboard sets the operational mode of the INS, uses the function Insert to set the INS status word, and in the case of NAV mode sets the start time for navigation based on the system clock. A flag is also set to allow the program to enter the function Mode_Change_Timer when an operational mode changes. When the ESC key is struck the exit flag is set and on return the simulation will terminate.

In the case of I01 and I24 the function processes each character separately and completes operation based on a CR. Valid_Msg_Num is utilized to check the validity of these inputs. If valid the number is accepted. If invalid the computer bell sounds and an invalid message is printed to screen.

An integer value is returned from this function. The input status (CR or no CR), message id (D or I), and the message number are OR'd together and this integer is analyzed by Msg_Cmd_Proc to determine keyboard inputs. Note that single key inputs such as ESC are not interpreted directly by Msg_Cmd_Proc but rely on flag setting for their interpretation.

E.1.18 Matrix_Ops

Host Program: INS_LN94.C
Calling Function(s): Derivative
Noise

Calling Convention:

```
Matrix_Ops(char,double[][],int,double[],int,double[],int)
```

Matrix_Ops is called to multiply two matrices together. Permissible matrices are a two dimensional matrix post multiplied by a

one dimension vector. The operation type is passed to the function and the function uses a switch decision loop to select the operation. If a request other than "*" is requested the function and program will terminate. The two dimensional matrix and its column dimension, the single dimension vector and its row dimension, and the resultant matrix and its row dimension are passed to the routine. The function initializes the resultant matrix to zero and then does a row by column multiply of the two matrices.

E.1.19 Mode_Change_Timer Host Program: INS_LN94.C
Calling Function(s): Msg_Cmd_Proc

Calling Convention: Mode_Change_Timer()

The simulation does not perform alignments directly. Instead a timing loop is initiated to simulate alignment. This routine is entered from Msg_Cmd_Proc anytime that GC align, SH align, or INFLIGHT align are called for from the keyboard. The function uses a switch decision loop to set the start time relative to simulation time, and sets the countdown flag which indicates countdown in progress. It also changes the mode on the simulation screen.

E.1.20 Model1 Host Program: INS_LN94.C
Calling Function(s): Derivative

Calling Convention: Model1()

This function is called by Derivative and it provides an updated version of the system dynamic matrix and process noise matrix for use in the next integration step. This function is called for the computation of each Kutta-Merson equation.

The function first equates the model variables with the latest trajectory data, and then computes the vertical gains. The ECEF-to-true-frame transformation and the sensor-to-true-frame transformations are computed. Elliptical gravity, craft rate, earth rate, platform angular rate, earth radius, and linear velocity magnitude are also computed based on the trajectory data. Non-zero components of the 23-state dynamic matrix are updated as well as the components of the process noise matrix.

E.1.21 Msg_Equal

Host Program: INS_LN94.C
Calling Function(s): Process_CMD_2
Update

Calling Convention: Msg_Equal(unsigned[],unsigned[],int)

This function is called to help reduce the computation overhead by not computing if the message of concern has not changed. The function does a component-by-component comparison of two messages or arrays and returns a FALSE if components are different, else a TRUE is return. The first difference that is encountered will return a FALSE, which eliminates the need of checking the entire array.

E.1.22 Msg_Cmd_Proc

Host Program: INS_LN94.C
Calling Function(s): Main

Calling Convention: Msg_Cmd_Proc()

This function is called by Main to handle any inputs from the keyboard. It processes the return from Keyboard, and if a CR has been input, the function will extract the ID of the message requested and its number. Once identified, the message number requested is updated to the screen. In addition, if the request is for the input message,

Write_Data is called to update the screen. The function Update handles all output messages from this point.

The function is used primarily to interpret and action message changes. However, if key input is a function key requesting a mode change other than NAV, the function will call Mode_Change_Timer to start the alignment timing process.

E.1.23 Noise

Host Program: INS_LN94.C
Calling Function(s): Propagate

Calling Convention: Noise()

This function is called to provide the white Gaussian noise vector that is injected into the system model. The function calls Random_Number to generate a vector of Guassian random numbers. Since the process noise matrix is diagonal, it is not necessary to make any computations such as Cholesky square root and since G is identity, the approximation that this function would normally implement, $Q_d = GQG\Delta t$, now becomes $Q_d = Q\Delta t$. Delta t comes from Integrate where this time represents the last time Q_d was computed. Utilizing Matrix_Ops, the function post multiplies Q_d by the Guassian random number vector and the result is a vector with statistics $N[0,Q_d]$.

E.1.24 Plot_Results

Host Program: SETUP.C
Calling Function(s): Main

Calling Convention: Plot_Results()

This function is called at the end of the simulation to give the user the opportunity to plot any data that may have been collected during the simulation. The plotting routine, EZPLOT, is invoked when indicated by the user. If the user chooses to plot results on return

from EZPLOT, the function will query the user as to whether to restart the simulation or exit the program. If plotting is not desired, the user may restart the simulation or exit the program.

The function uses Essential Software libraries to manipulate the screen, and formatted input/output to query the user.

E.1.25 Process_CMD_1 Host Program: PROCESS.C
Calling Function(s): Main

Calling Convention: Process_CMD_1()

This function provides the MILSTD 1553B formatted data that is output to the bus controller. It provides the response for command word #1 of the FNU 85-1 specification. Refer to Appendix A for the words and formats that are transmitted by the simulation.

The function provides processing of word 1 of the response to command word #1, which is the INS status word. If the mode selected is an alignment mode the function sets the appropriate bits to indicate the mode. In addition, it monitors the progress of the alignment and displays ATT RDY, DEG RDY, and NAV RDY as time progresses.

All subsequent words are processed by scaling the INS-indicated solution from INS_Error_Include according to the FNU 85-1 specification and outputting the results as an integer in hexadecimal form.

E.1.26 Process_CMD_2 Host Program: PROCESS.C
Calling Function(s): Main

Calling Convention: Process_CMD_2()

Process_CMD_2 is called by Main in response to command word #2 of the FNU 85-1 specification. There are only four words, msp and lsp of present position latitude and longitude. The function utilizes

Msg_Equal to check whether the position has changed since the last process. If the position has changed the words in CMD_1 and CMD_2 are set to new position. Display_Lat_Long and Write_Data are called to display the results to the screen.

E.1.27 Process_CMD_24 Host Program: PROCESS.C
Calling Function(s): Main

Calling Convention: Process_CMD_24()

In a similar fashion to Process_CMD_1, this function processes the applicable words in response to command word #24 of the FNU 85-1 specification. It does not specifically process the INS status word but sets the status word for this function equal to the status word of the Process_CMD_1.

E.1.28 Propagate Host Program: INS_LN94.C
Calling Function(s): Main

Calling Convention: Propagate(double)

Propagation is the coordinating function for propagation of the system model. The function passes end time to and calls Integrate. After Integrate returns, integration status is checked for integration error. If there is an error, Propagate terminates the program and an error message is displayed to the screen. If no error, and if at end time or at end step, the function computes Δt and calls Noise for process noise injection into the model. After noise injection, the state vector is set to the solution vector in preparation for the next integration cycle. This process continues until at end time, at which time Propagate returns to Main. Refer to Section 3.3 for a more detailed discussion of this function.

E.1.29 Random_Number

Host Program: INS_LN94.C
Calling Function(s): Noise

Calling Convention: Random_Number(int,unsigned int,int,double[])

Random_Number provides a vector of QNUM random numbers to the function Noise. It is seeded by system time at simulation start, and a vector of random numbers of quantity SIZE is generated. A random number is then generated to select one number from this vector. The function then replaces the selected number with another random number.

Each number of the vector passed back to Noise is the sum of twelve random numbers with 6.0 subtracted from that sum. This process gives a vector of approximately Guassian numbers with statistics of N[0,1].

E.1.30 rem_hdlr

Host Program: CLKT10.ASM
Calling Function(s): Main

Calling Convention: rem_hdlr()

This function removes the interrupt handler installed by init1553 and replaces it with the interrupt handler present before initialization of the simulation.

E.1.31 test_1553

Host Program: SETUP.C
Calling Function(s): main

Calling Convention: test_1553()

This function verifies the presence of a Ballard PC1553-2 interface board in the host computer. It writes test values to several locations in the on-board dual-port memory. It then reads them back to verify their values. If the values are not consistent, the function

assumes no board is present. If no board is present the function returns FALSE.

This function uses various library routines from the Ballard PC1553 interface program. Readers are referred to Ballard PC1553 User's Manual [5] for details on these functions.

E.1.32 Trajectory

Host Program: INS_LN94.C
Calling Function(s): Derivative

Calling Convention: Trajectory()

Function Trajectory is called by Derivative and provides the updated trajectory data so that the system model can be updated prior to computing its derivative.

For the static navigation case, the trajectory data is unchanged from the start to the end of the navigation period. Therefore, the data is coded into the function and is accessed if static navigation is requested. If a flight trajectory is requested, function Introduction opens the applicable trajectory data file for reading and the data is read from the file by Trajectory.

E.1.33 Update

Host Program: INS_LN94.C
Calling function(s): Main

Calling Convention: Update()

Update is used to keep the data on the simulation screen current. It updates the operational mode and status of alignment; ATT RDY, DEG RDY, and NAV RDY. It also uses Display_Lat_Long to update present position lat and long and displays simulation time. Msg_Equal is used to minimize the computation load when the present position has not change. Update then uses a switch decision loop to determine the output

message and calls Write_Data to keep the screen data of the message current.

E.1.34 Valid_Msg_Num

Host Program: INS_LN94.C
Calling Function(s): Keyboard

Calling Convention: Valid_Msg_Num(int,int)

Valid_Msg_Num is called by Keyboard to validate keyboard input when the user tries to change the messages on the simulation screen. The message ID and the message number are passed to the function. If the corresponding ID and number are valid, TRUE is returned to the calling function, else FALSE is returned. ID and number must coincide. In other words a user can not input D24 versus I24. Each component of the message ID and number is confirmed as it is entered so the program must include all valid numbers. For example, in I24 the ID of I and the number 2 and 24 must be valid inputs.

E.1.35 Write_Data

Host Program: INS_LN94.C
Calling Function(s): Display_Screen
Initialize
Process_CMD_2
Update

Calling Convention: Write_Data(unsigned int[],int,int)

Write_Data prints the contents of input/output messages in the format that is established for the simulation screen. Only the modified words are updated.

The message array, the length of the array, and the type of message (input/output) are passed to the function. The type designates the cursor position when preparing to update. The length sets the duration of the update cycle. All the messages vary in the number of

words and it makes no sense to update beyond message length. The function sets the position of the cursor and then outputs a hexadecimal number to the appropriate position on the screen.

E.2 Commercially Developed Functions

The following is a summary of the functions utilized from commercially available software packages. As mentioned previously, details are not provided for these functions except to say what they do and indicate where to get more detail.

E.2.1 Essential Software C Utility Library

This library, SLIBES.LIB, is linked with the program INS_LN94.C and the following functions were utilized:

1. border--set screen border to specified color
2. clreol--clear to end of current line
3. clrscrn--clear current screen
4. clscolor--set current screen to specified color
5. colptrc--print character in specified color
6. colrptrf--print a string in formatted mode
7. colrprt--print a string in specified color
8. curlocat--locate the cursor by row and column

The reader is referred to the Essential Software C utility Library reference manual [11].

E.2.2 Ballard PC1553 Functions

The following is a list of the functions utilized from the Ballard Technology PC1553 software:

1. readrtdwd--reads data from a PC1553 sub-address

2. writertdwd--writes data to a PC1553 sub-address

3. config--configures PC1553 by writing to the three control registers

The reader is referred to the Ballard PC1553 Interface User's Manual [5].

Appendix F: Simulation Operator's Guide

This appendix will summarize system/hardware requirements and the procedural steps to be taken when operating the LN-94 INS Simulation program. The intent of this guide is not to outline the operation of peripheral devices or operating systems. It is assumed the user has a working knowledge of an IBM compatible personal computer and the DOS operating system. It is also assumed that the user is familiar with the operation of the DTI bus control software and the physical requirements of the MILSTD 1553 bus connections. It is suggested that the user also become familiar with the EZPLOT plotting routine User's Manual.

This guide will cover the computer system requirements, hardware requirements, and the required configuration of the bus controller. A step-by-step setup and operational procedure of the simulation is provided.

F.1 System Requirements and Hardware

As indicated in Chapter 3, the simulation program is compiled to take advantage of a 80387 or compatible coprocessor. In addition, the system assumes a color monitor as the program addresses the video card for screen color manipulation. The minimum system requirements are, a 80386 computer with an 80387 coprocessor, and EGA/VGA color monitor. To maintain the speed of the program the processor should be at least 20Mhz and the system should have a hard drive. If no hard drive is present and a floppy drive must be used, then the user should ensure that data

storage rates are slow enough so as not to impact on the operation of the program.

The program executable is approximately 50K in size, and as long as sufficient RAM is available for the executable and a 7K stack, the program should operate without problem. For storage of data, a typical file in ASCII format, at a collection rate of 10 seconds for three hours, uses approximately 70K of memory. The user should have some idea of storage requirements prior to starting the simulation.

The simulation requires a Ballard PC1553-2 interface board installed on the host computer bus. The DTI bus controller software is to be installed in another PC, and the two computers interfaced via the Ballard boards and the appropriate 1553 bus cabling and couplers. If desired a printer can be interfaced to the program host computer to take advantage of the plotting routine EZPLOT. EZPLOT only has drivers for dot matrix printers and a good 24-pin dot matrix printer should be sufficient for quality output. If a printer is not available, EZPLOT outputs to the screen and files may also be transported to another system for printing.

F.2 Simulation Setup

Where the executable is located on disk is not particularly important. The only directory requirement is that the program must be able to access a directory path, C:\UTILITY\EZPLOT, to store data files. The plotting routine, EZPLOT.EXE, as well as the two trajectory files, STRAIGHT.TRJ and FIGHTER.TRJ, must also be present in this directory.

Boot the program host computer and go to the simulation directory. At the prompt type INS_LN94. The next screen the user sees will have the program title, version number, and programmer name. The program will prompt the user by the question:

DO YOU WISH TO READ THE PROGRAM OVERVIEW? Y/N

The user enters Y(y) or N(n). It should be noted here that any required inputs to screen prompts after this prompt must be follow by a CR. If the user chooses to see the overview, three screens of literature will be displayed giving the user a basic idea of the simulation capabilities and operating procedures. If the user chooses not to see the overview or following the overview, this prompt will appear:

SELECT THE TRAJECTORY YOU WISH TO USE:
(1) LITTON FIGHTER TRAJECTORY (TWO HOUR SIMULATION)
(2) STRAIGHT AND LEVEL TRAJECTORY (MAX 6 HOURS)
(3) STATIC NAVIGATION FROM AFIT (NO TIME LIMIT)

Selection of 1 or 2 will open the applicable trajectory file for reading. Selection of 3 will set the navigation parameters to a stationary condition. The next prompt will be:

DO YOU WISH TO TRACK ERROR STATES OR WHOLE STATES:
(1) ERROR STATES
(2) WHOLE STATES
(3) NO STATE TRACKING

Based on which states are selected, the user will see a list of those states which will be tracked and stored. If no states are tracked, this will be indicted on the screen as well. Type a CR and the next prompt will be:

THE LENGTH OF YOUR NAVIGATION SESSION IN MINUTES IS:

The user inputs an integer representing the length of the session in minutes. When making selection the user should remember the type of

trajectory that was selected and the stipulated length of the trajectory. The next prompt is:

```
IF YOU SELECTED STORAGE OF DATA, ENTER DATA COLLECTION
RATE IN SECONDS ELSE ENTER ZERO:
```

The user enters an integer value for time between data collection points.

The next screen will give the user the opportunity to change any of the previously selected parameters, carry on with the simulation, or exit the simulation.

```
DO YOU WISH TO CHANGE ANY OF THE PARAMETERS YOU JUST SELECTED:
            (1)  TRAJECTORY
            (2)  STATES
            (3)  SESSION LENGTH
            (4)  TIME BETWEEN COLLECTION POINTS
            (5)  NO CHANGE
            (6)  EXIT PROGRAM
```

The user inputs the appropriate request. If the user chooses to change a parameter, only that parameter prompt will be displayed and then returned to the above screen. If the user chooses to exit the program the DOS prompt will reappear. To carry on with the simulation enter the selection for NO CHANGE.

F.3 Simulation Operation

When the user enters the simulation, the screen will change to the formatted display necessary to show the information applicable to the simulation. The present position should be zero latitude and longitude and the input and output message should show zeros. At this point the user should setup the 1553 bus before carrying on with any simulation operation.

The user should establish bus control with the DTI bus controller software, Sceptre [13]. Set the bus controller to send present position latitude and longitude to the simulation. The input is in accordance with command word #2 in FNU 85-1 [3]. Refer to Table A.1 for details of all the command words that the simulation responses to. The RT address of the INS is 05h. The process is master to remote, sub-address 02h, receive, and four data words. For present position the data words are:

1C4Ah	msp latitude
1B00h	lsp latitude
C435h	msp longitude
1F00h	lsp longitude

These represent the latitude and longitude of AFIT. Transmit the command once, and the simulation screen should now read the AFIT present position.

Return to the simulation and select one of the alignment modes as per the list presented on the simulation screen. The system MUST be aligned. If the user selects NAV before alignment, the NAV mode will be displayed but the system will not propagate. After alignment (approx 10 seconds), the user may enter the NAV mode. The simulation will enter the propagation mode and the input and output messages should begin to change.

At this point, the user may wish to display results of the simulation on the 1553 data bus. The two output messages that the simulation will respond to are in response to command word #1 and command word #24 both of which are outlined in Appendix A. Using the parameters in Table A.1, the user can set the bus controller to receive the data from the simulation.

As the navigation proceeds the simulation will store results to the EZPLOT directory. The simulation will navigate until the end of the navigation session or until the user manually exits the program using the ESC key. When the navigation session is terminated at the end of the navigation time the following prompt is presented to the user:

YOUR XX MINUTE NAVIGATION SESSION IS COMPLETE

or if the simulation is terminated by the user, the following is presented:

SIMULATION HAS BEEN TERMINATED PRIOR TO THE END OF NAV SESSION

In either case, the system will wait for a CR and then display the following:

DO YOU WISH TO PLOT ANY OF YOUR RESULTS:
(1) YES
(2) NO EXIT PROGRAM
(3) NO RESTART SIMULATION

If the user selects 2 the program will terminate and the DOS prompt will reappear. If the user chooses to restart the simulation, the sequence in Section F.2 will start again. If plotting is selected the simulation will turn control over to the routine EZPLOT. The user should refer to the user's manual of EZPLOT to produce the plots desired. The routine is self-explanatory and should not present any difficulties. For using EZPLOT the file names used in the simulation are listed in Table F.1.

Once the user terminates EZPLOT control is returned to the simulation and the following is displayed:

DO YOU WISH TO START THE SIMULATION AGAIN:
(1) YES
(2) NO EXIT PROGRAM

Table F.1 File Names Used by EZPLOT

Error States	File Name	Whole States	File Name
δL	del_lat.dat	longitude	long.dat
δl	del_long.dat	latitude	lat.dat
$\delta\theta_z$	d_thetaZ.dat	wander angle	wander.dat
ϕ_x	phi_X.dat	altitude	alt.dat
ϕ_y	phi_Y.dat	north velocity	vel_n.dat
ϕ_z	phi_Z.dat	east velocity	vel_e.dat
δV_x	vel_N.dat	vertical velocity	vel_v.dat
δV_y	vel_E.dat	north acceleration	acc_n.dat
δV_z	vel_Z.dat	east acceleration	acc_e.dat
δh	delta_h.dat	vertical acceleration	acc_v.dat
δh_c	del_h_c.dat	roll	roll.dat
		pitch	pitch.dat
		platform azimuth	azimuth.dat

If restart is selected, the sequence as described in Section F.2 will start again. Exit will return the user to the DOS prompt.

F.4 Summary

As can be seen, the simulation is straight forward and steps the user through the required inputs. By reading the program overview at the beginning of the program and studying the steps outlined in this guide, the user should have no problem operating the simulation.

Appendix G: 23-State Model Validation Results

This appendix provides a summary of the plots used to validate the 23-state model.

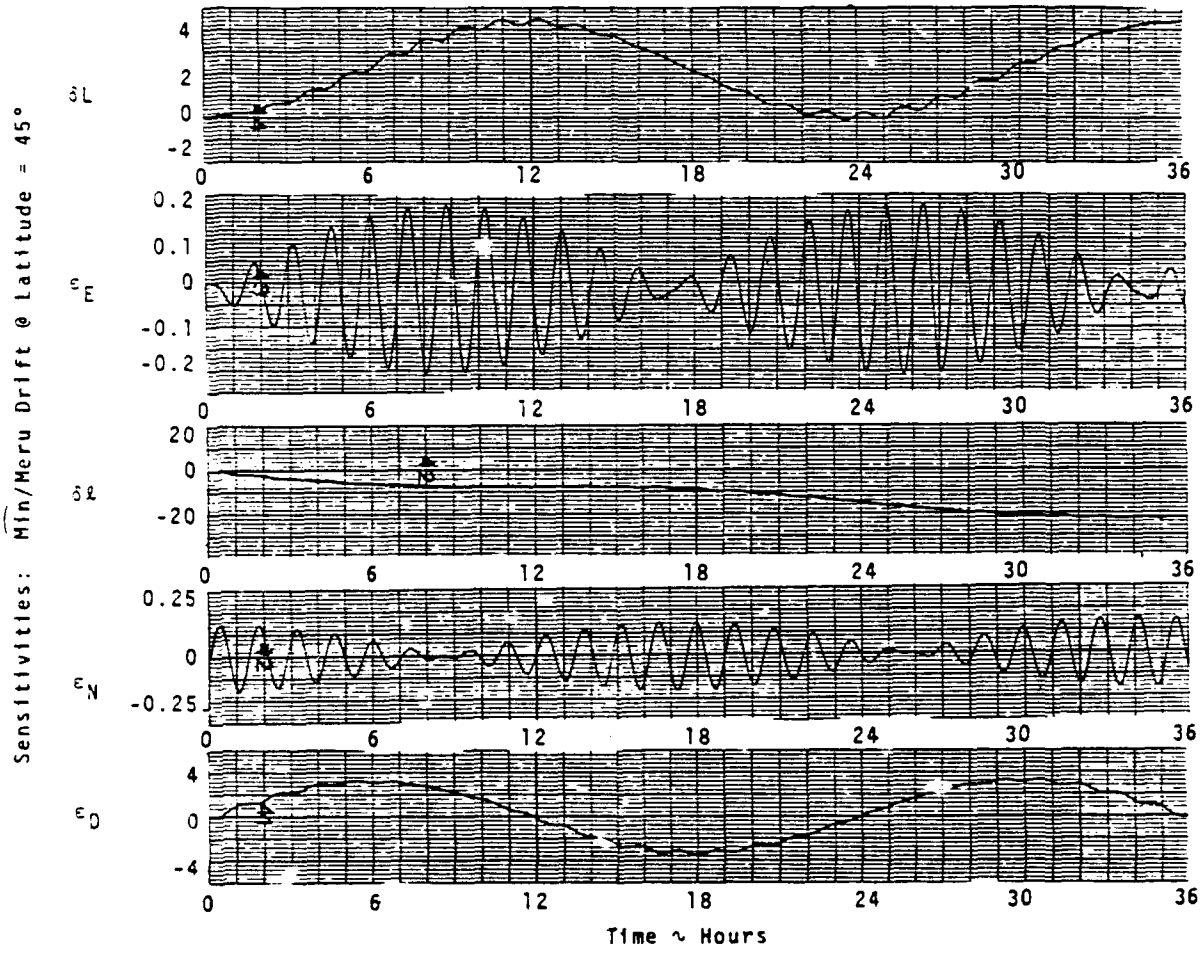


Figure G.1 Britting Results for North Gyro Drift

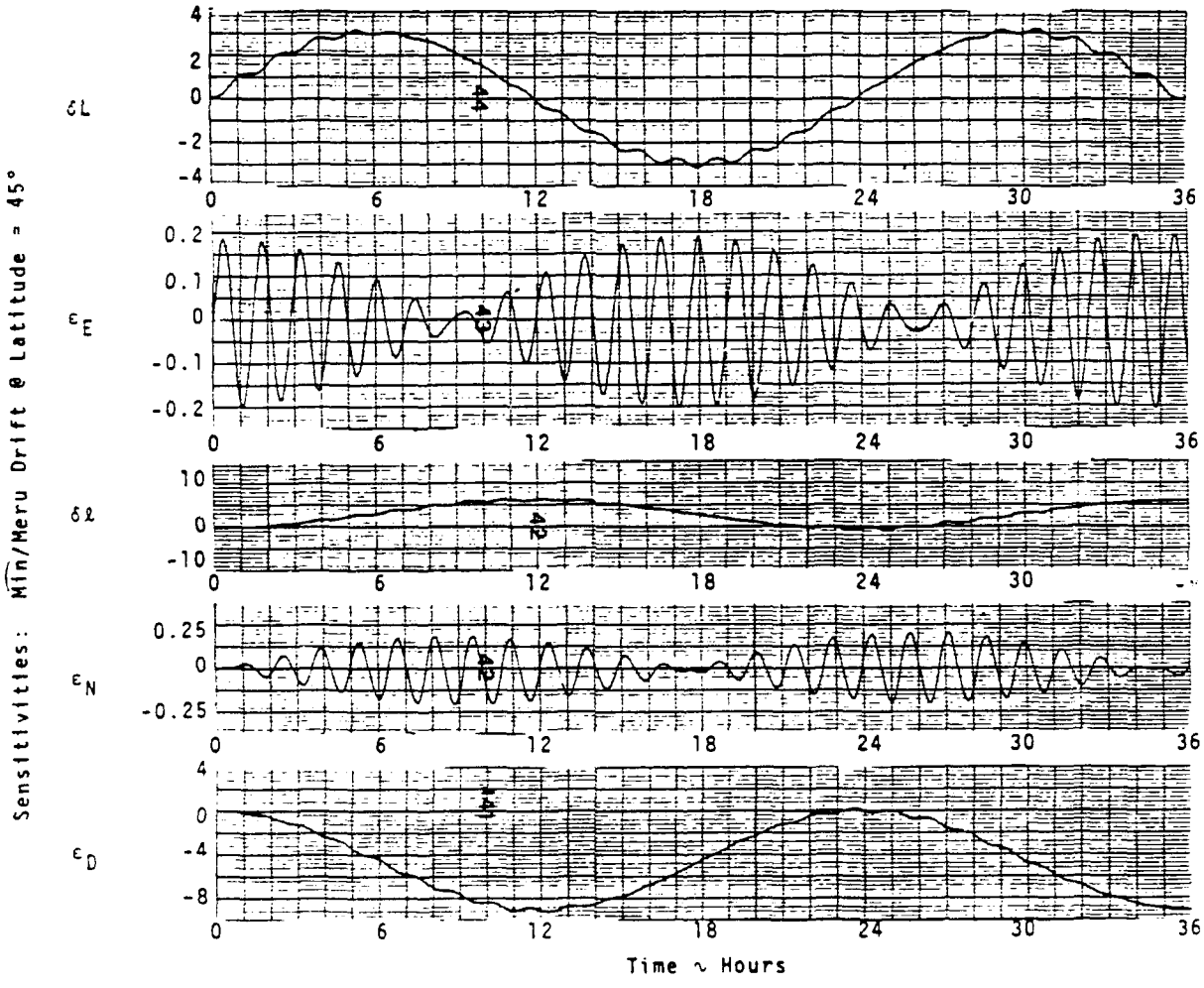


Figure G.2 Britting Results for East Gyro Drift

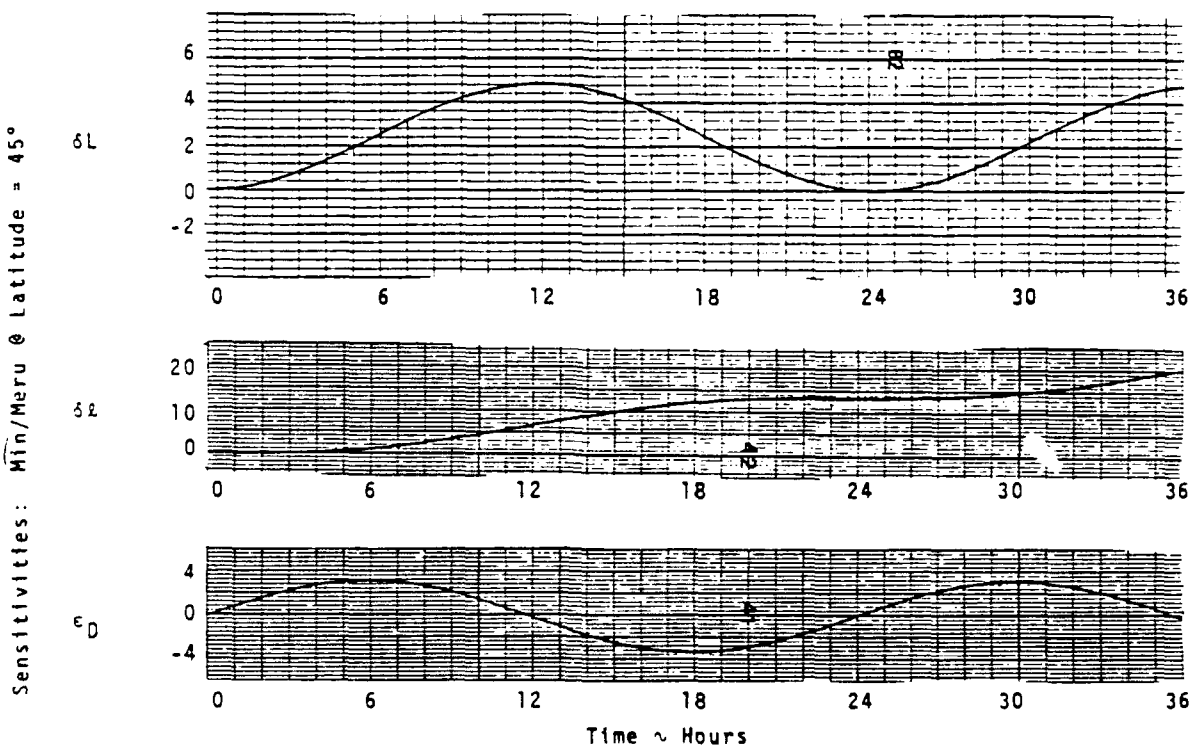


Figure G.3 Britting Results for Azimuth Gyro Drift

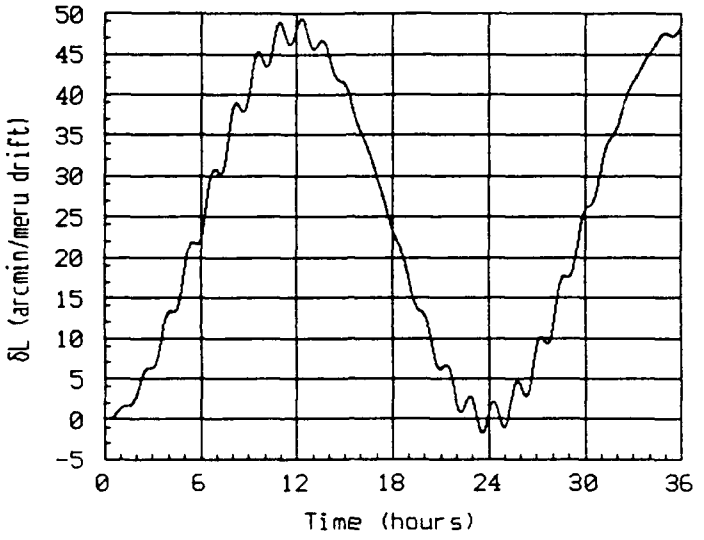


Figure G.4 δL for North Gyro Drift

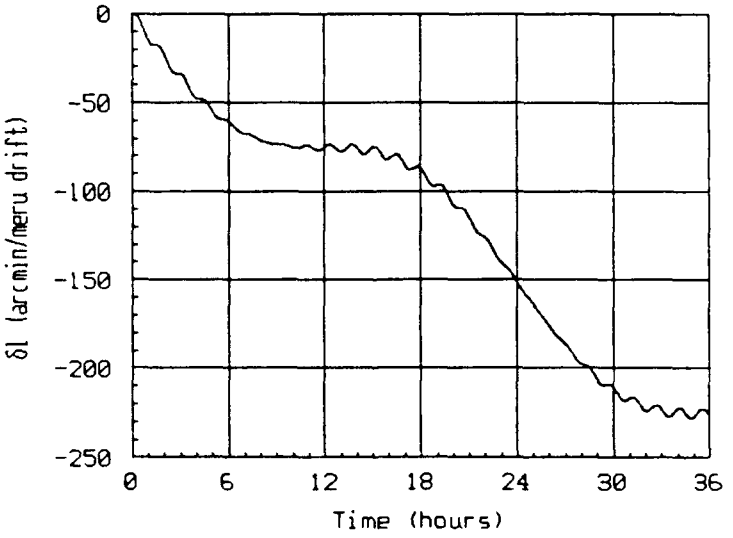


Figure G.5 δl for North Gyro Drift

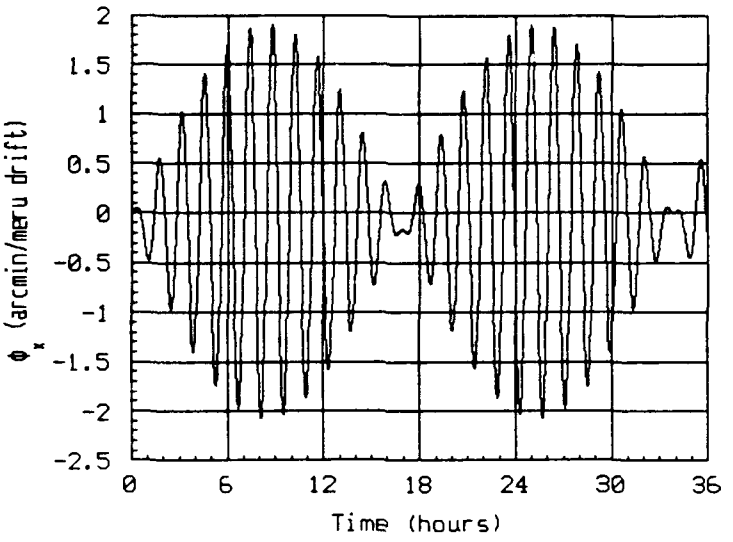


Figure G.6 ϕ_x (ϵ_E) for North Gyro Drift

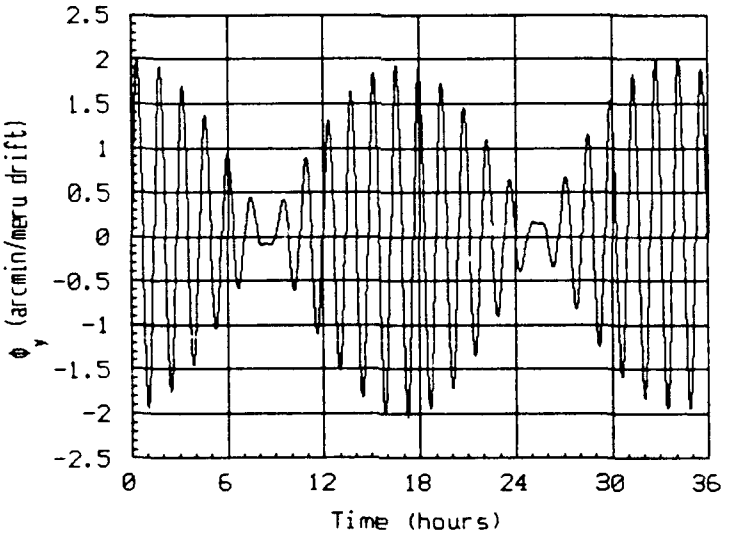


Figure G.7 ϕ_y (ϵ_N) for North Gyro Drift

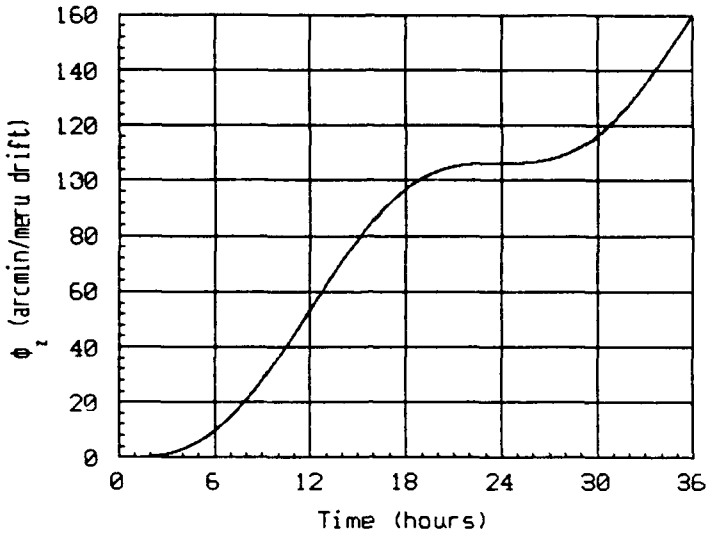


Figure G.8 ϕ_z ($-\epsilon_D$) for North Gyro Drift

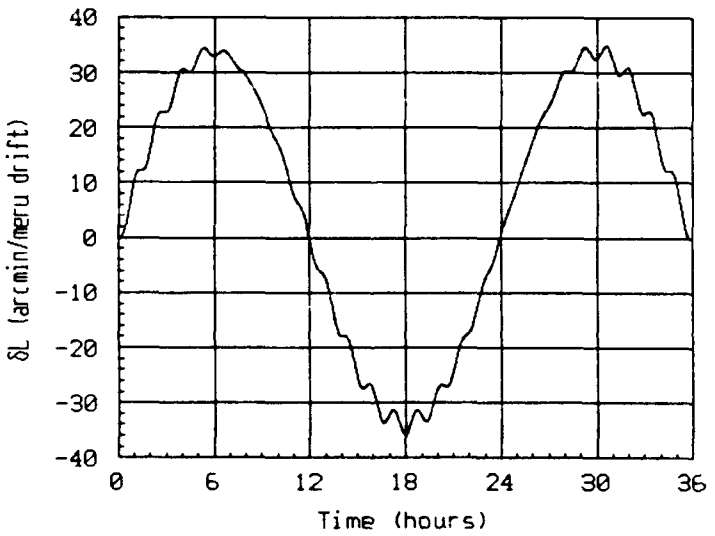


Figure G.9 δL for East Gyro Drift

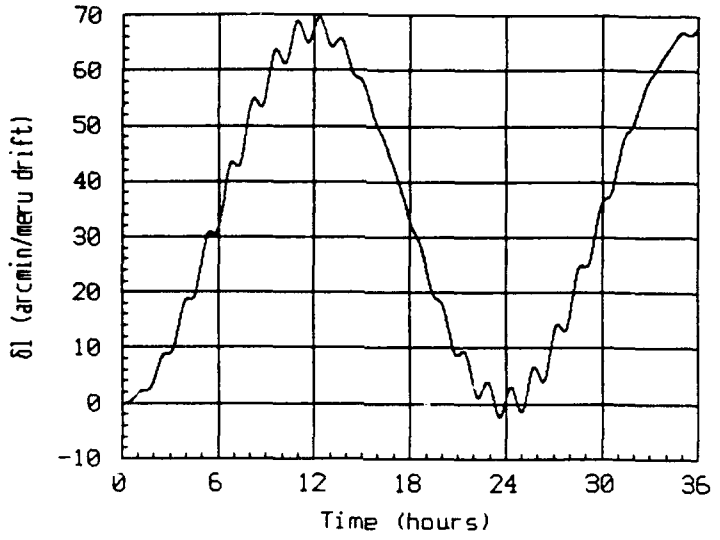


Figure G.10 δl for East Gyro Drift

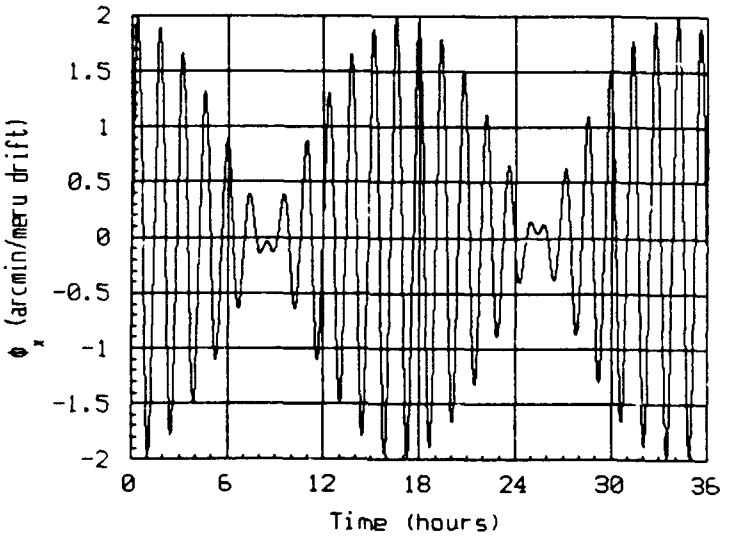


Figure C 11 ϕ_x (ϵ_E) for East Gyro Drift

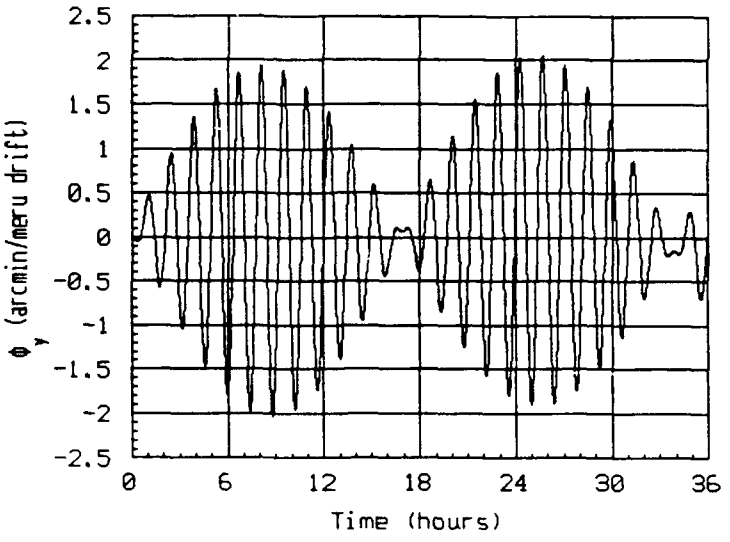


Figure G.12 ϕ_y (ϵ_N) for East Gyro Drift

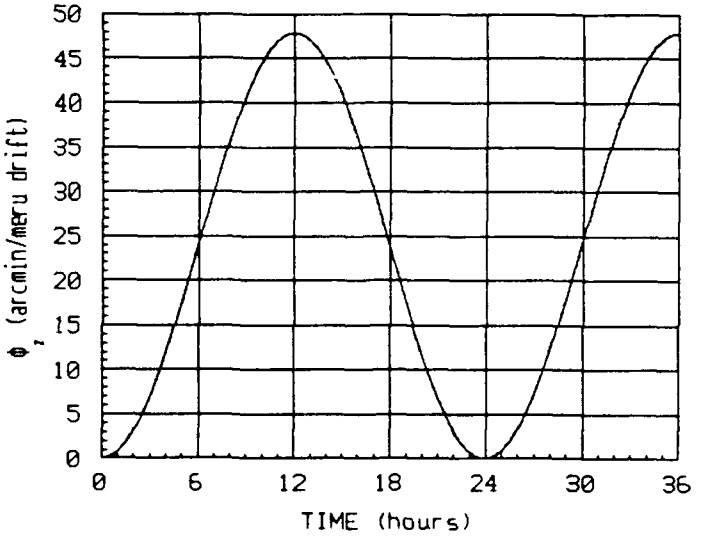


Figure G.13 ϕ_z ($-\epsilon_D$) for East Gyro Drift

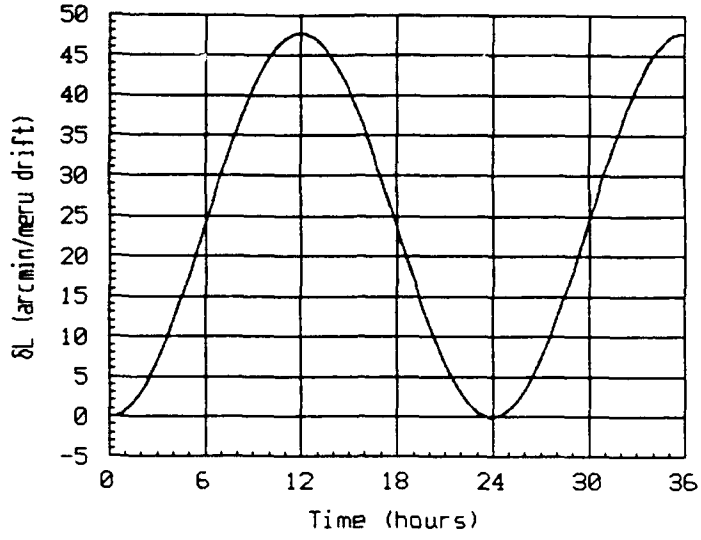


Figure G.14 δL for Azimuth Gyro Drift

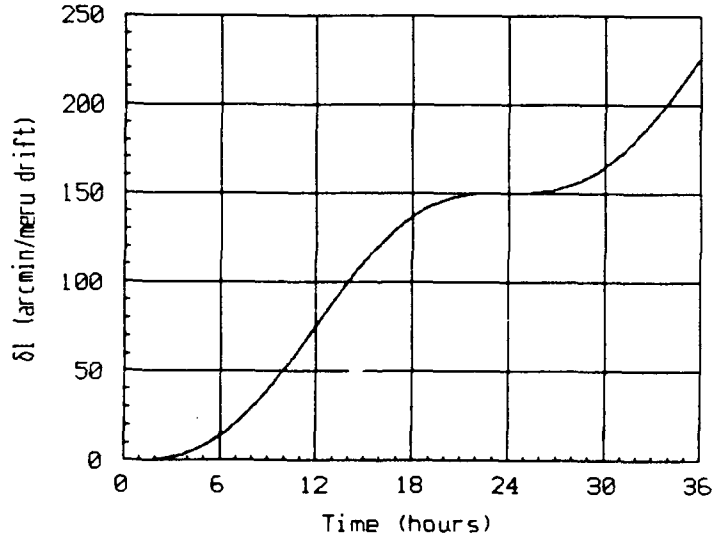


Figure G.15 δl for Azimuth Gyro Drift

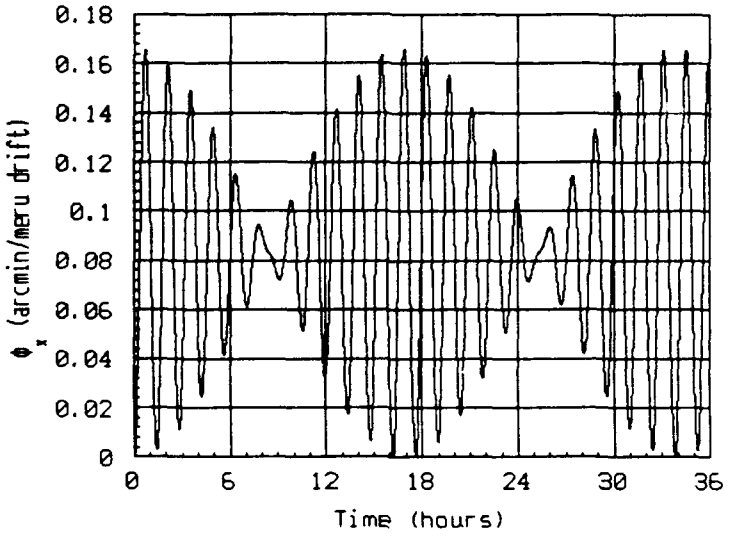


Figure G.16 ϕ_x (ϵ_E) for Azimuth Gyro Drift

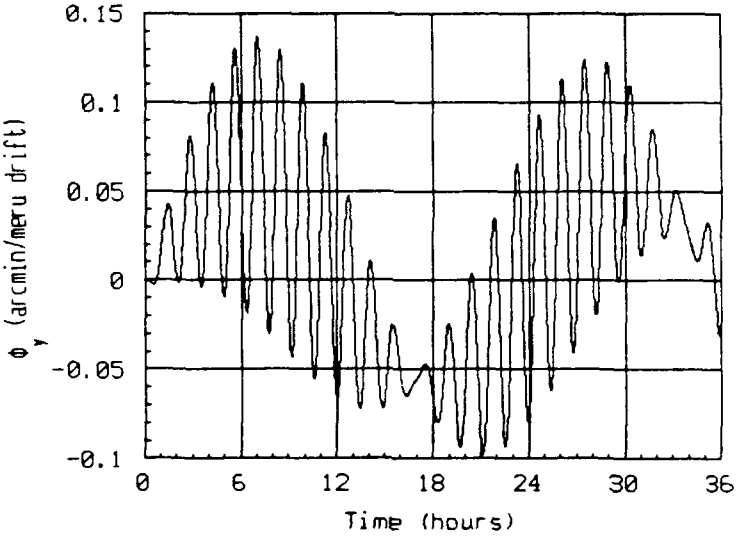


Figure G.17 ϕ_y (ϵ_N) for Azimuth Gyro Drift

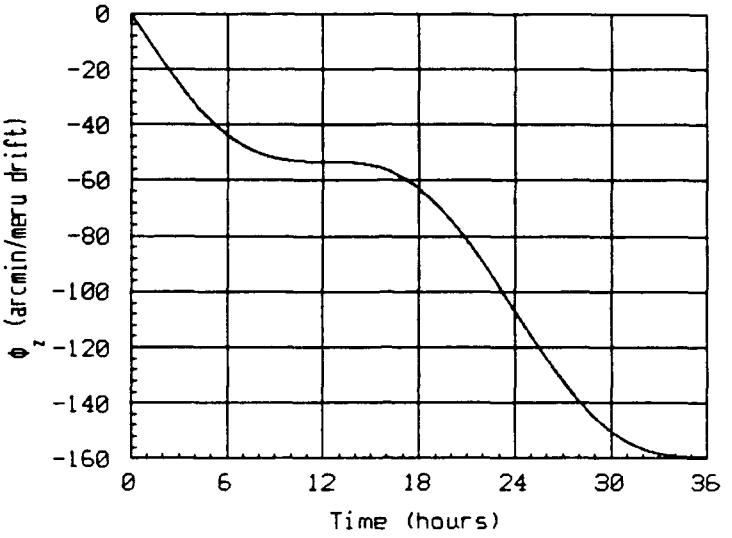


Figure G.18 ϕ_z ($-\epsilon_D$) for Azimuth Gyro Drift

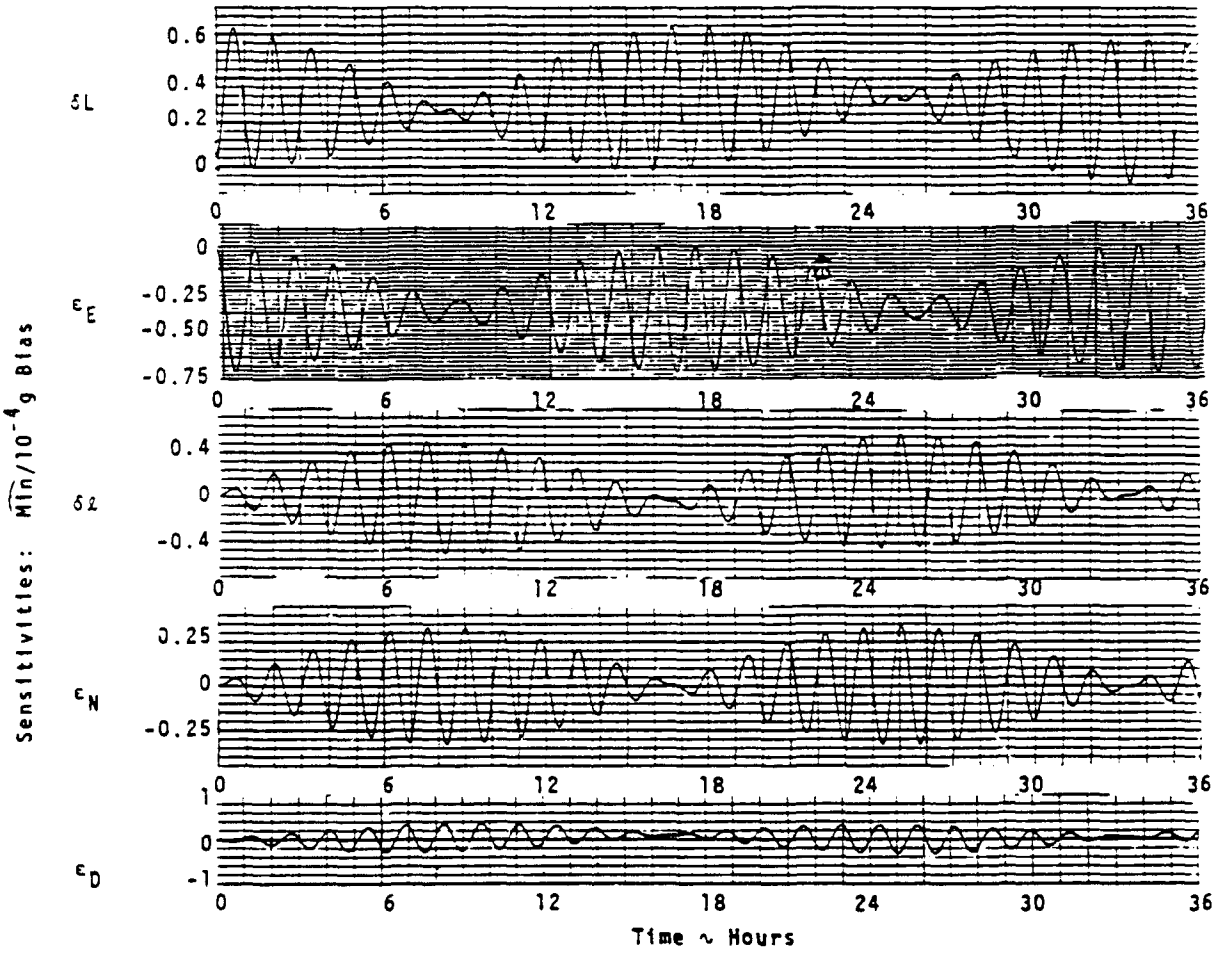


Figure G.19 Britting Results for North Accelerometer Bias

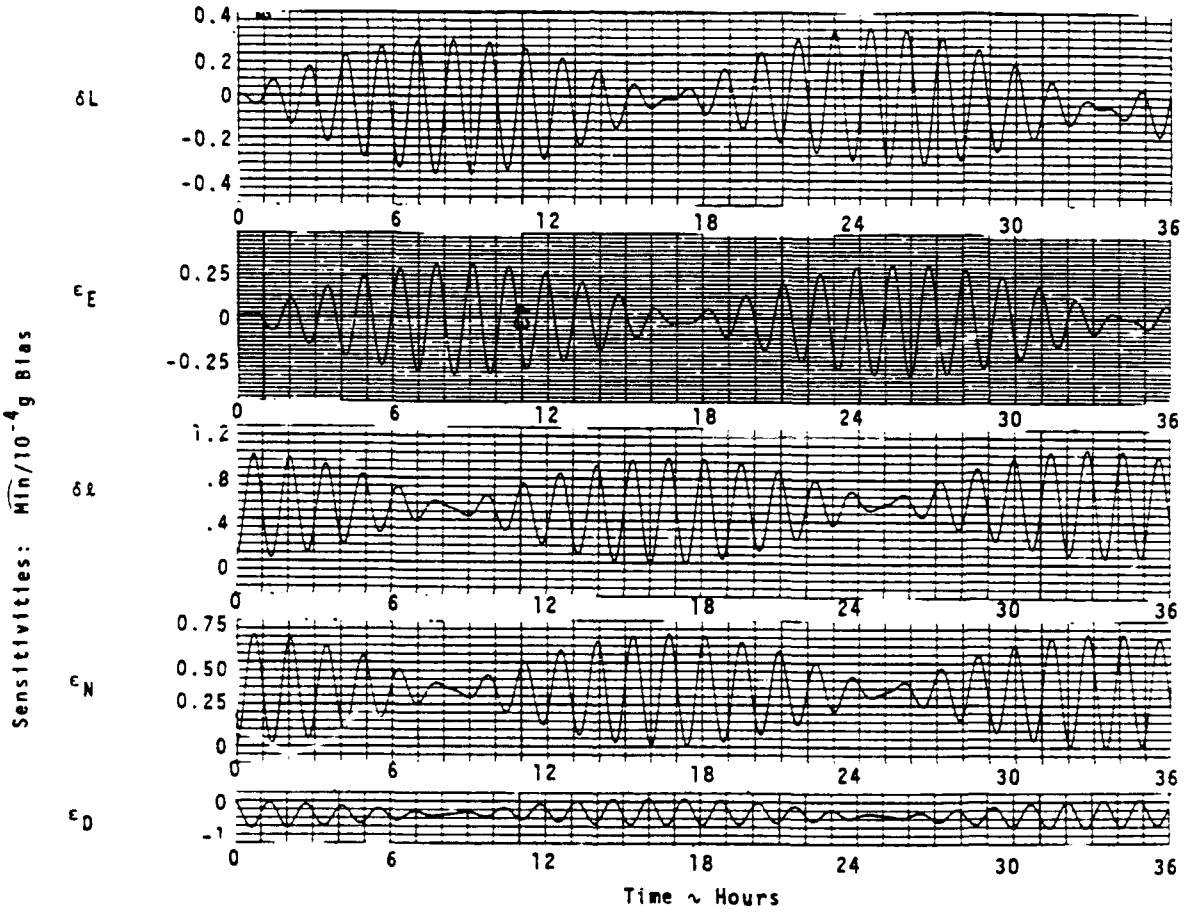


Figure G.20 Britting Results for East Accelerometer Bias

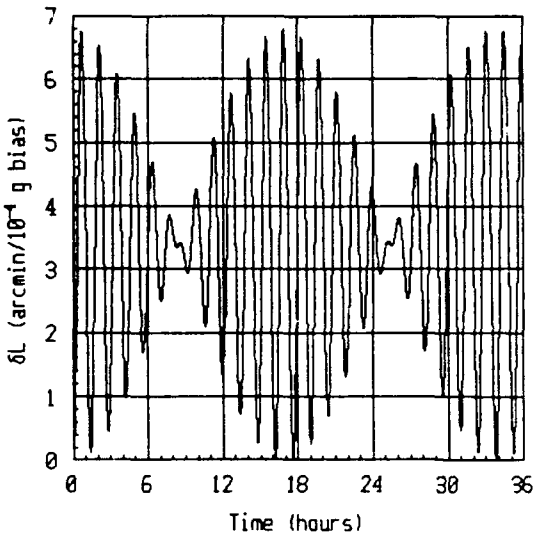


Figure G.21 δL for North Accelerometer Bias

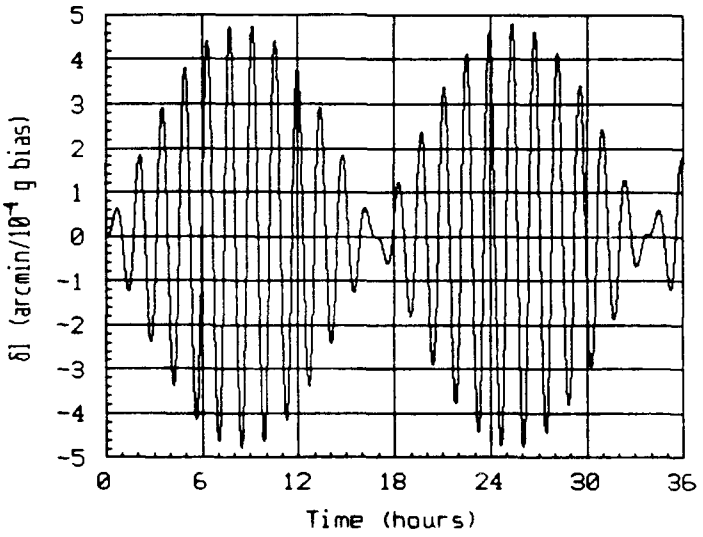


Figure G.22 δl for North Accelerometer Bias

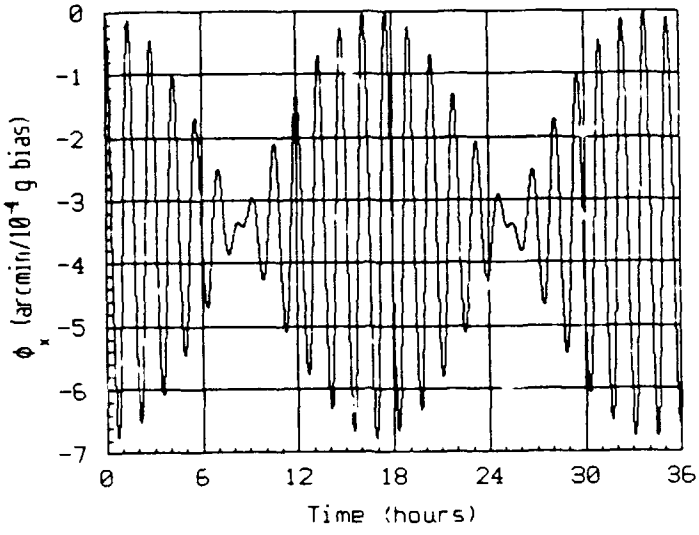


Figure G.23 ϕ_x (ϵ_E) for North Accelerometer Bias

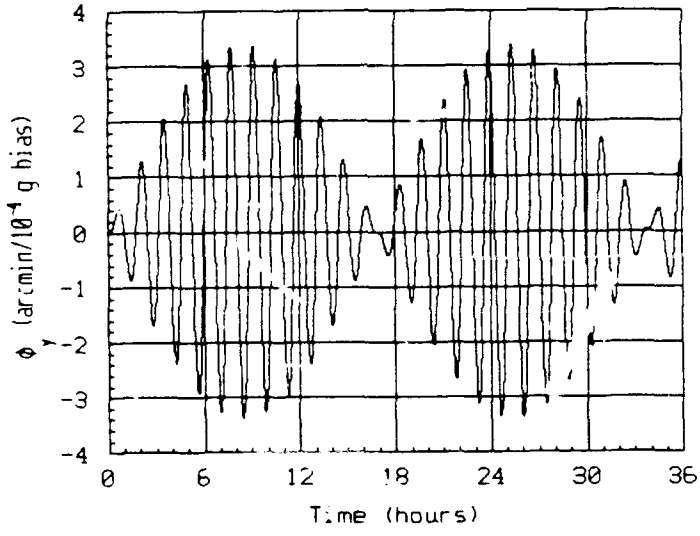


Figure G.24 ϕ_y (ϵ_N) for North Accelerometer Bias

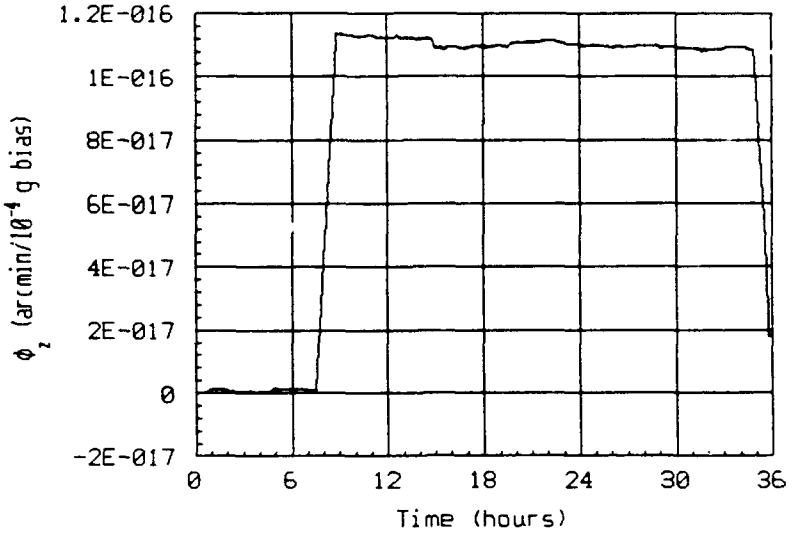


Figure G.25 ϕ_z ($-\epsilon_D$) for North Accelerometer Bias

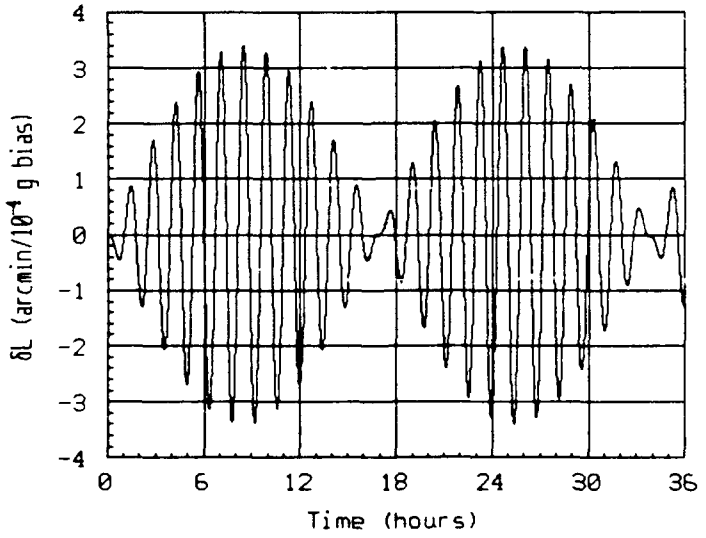


Figure G.26 δL for East Accelerometer Bias

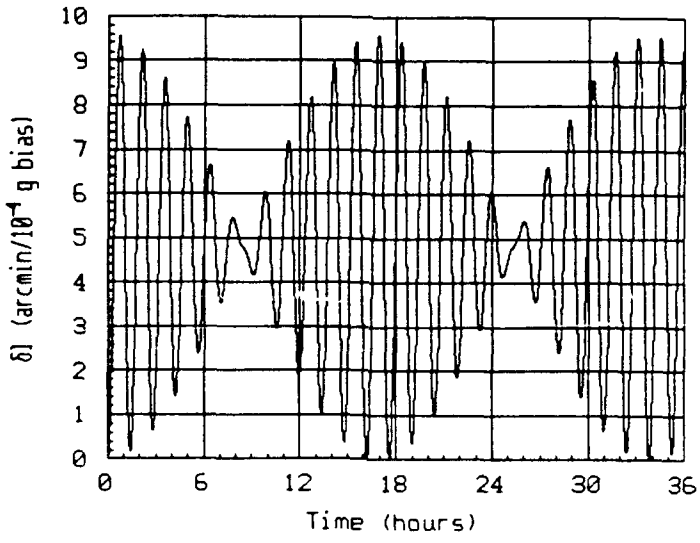


Figure G.27 δl for East Accelerometer Bias

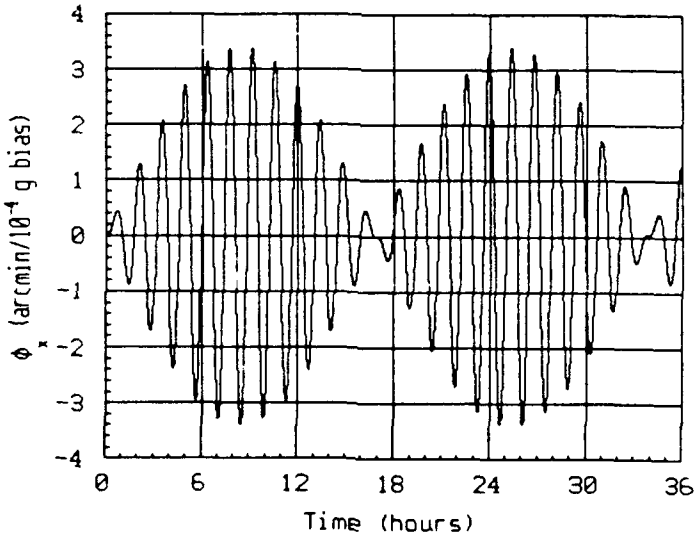


Figure G.28 ϕ_x (ϵ_E) for East Accelerometer Bias

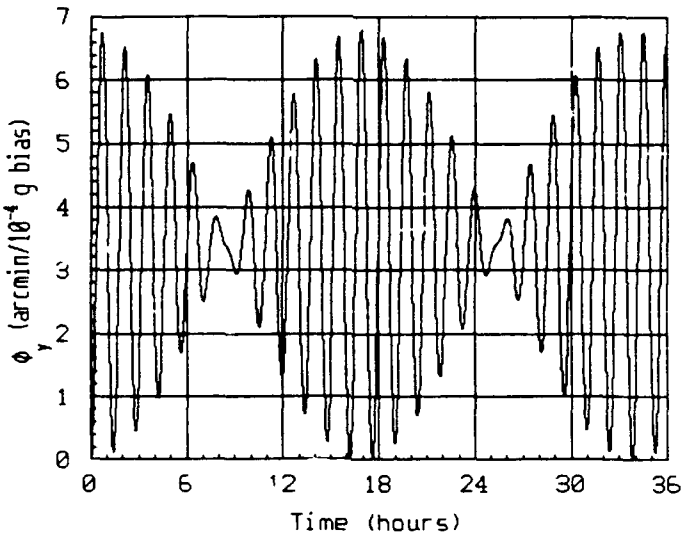


Figure G.29 ϕ_y (ϵ_N) for East Accelerometer Bias

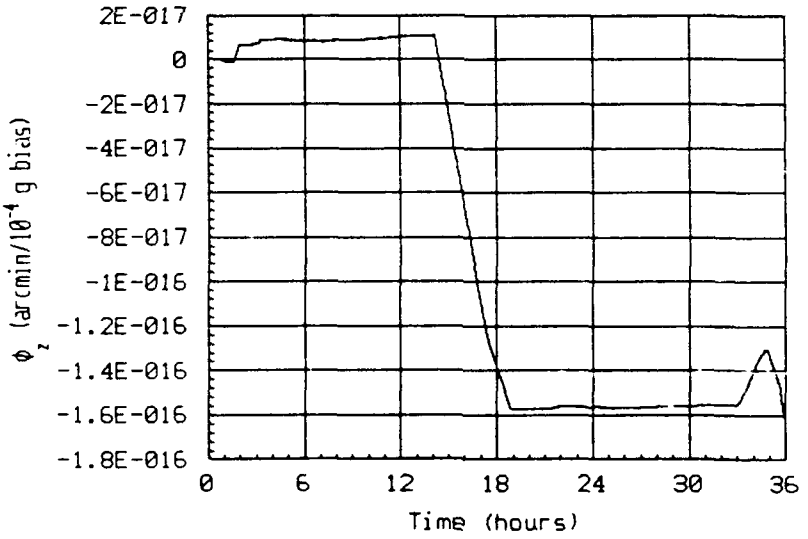


Figure G.30 ϕ_z ($-\epsilon_D$) for East Accelerometer Bias

Appendix H: Simulation Validation Results for Static Navigation

This appendix is a summary of the plots that were generated for the static navigation validation. All plots except Figure H.3, Figure H.11, and Figure H.12 show a comparison of the actual INS data and the data generated by the simulation. Figure H.3 shows the altitude results from the simulation. The altitude for the INS is zero because the baro-altimeter is not modelled and therefore has no uncertainty. Figure H.11 and Figure H.12 display plots of the level acceleration components experienced by the INS. Plots for the simulation are zero as accelerometers are not modelled in the simulation.

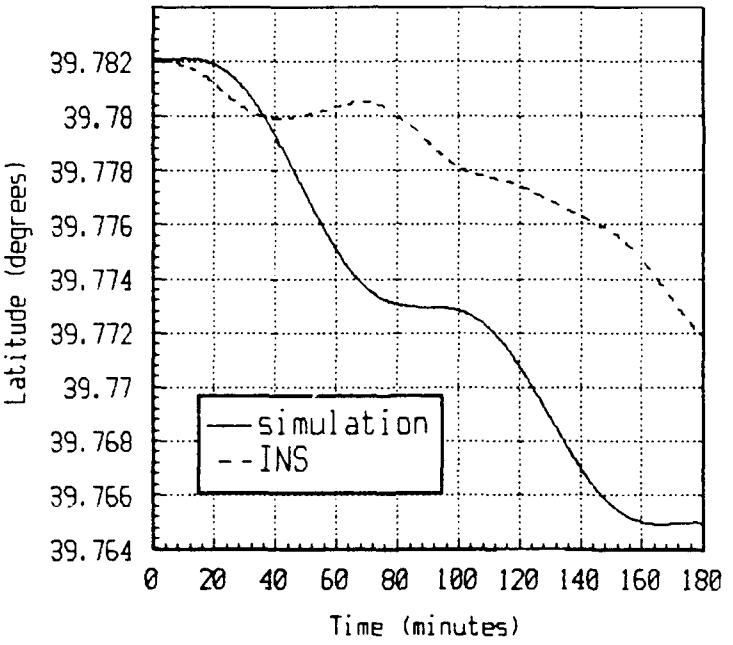


Figure H.1 Latitude for Simulation and INS-3 Hr Stationary Navigation

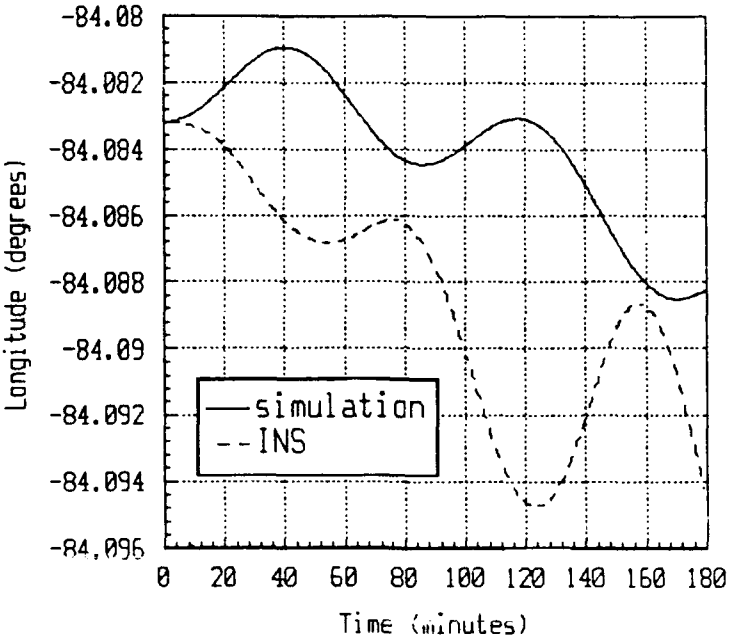


Figure H.2 Longitude for Simulation and INS-3 Hr Static Navigation

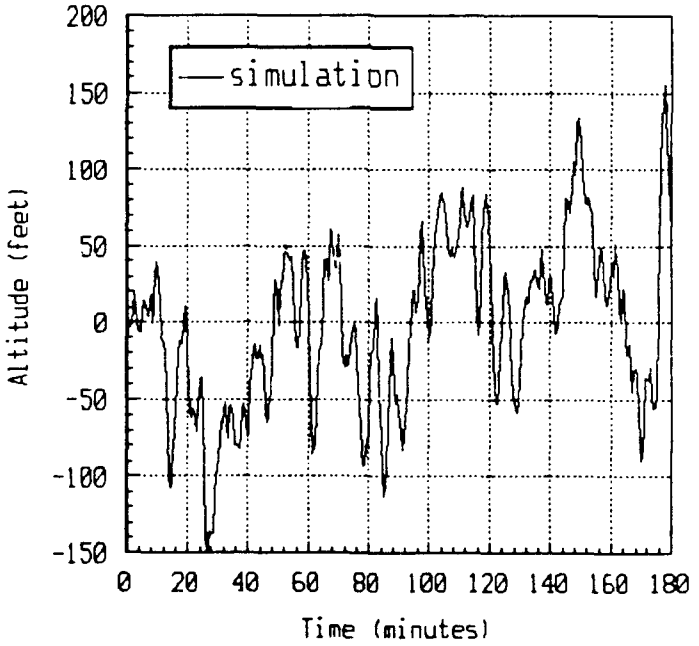


Figure H.3 Altitude from Simulation 3 Hr Static Navigation

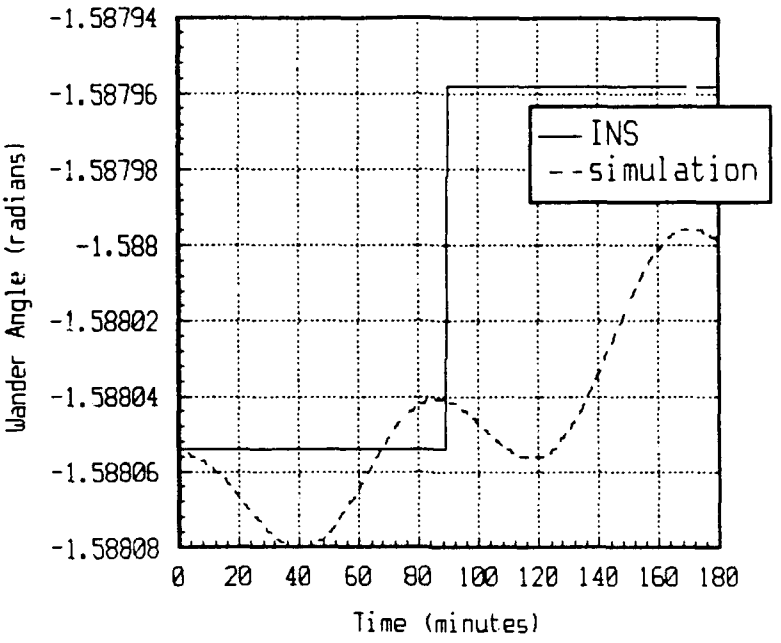


Figure H.4 Wander Angle From Simulation and INS-3 Hr Static Navigation

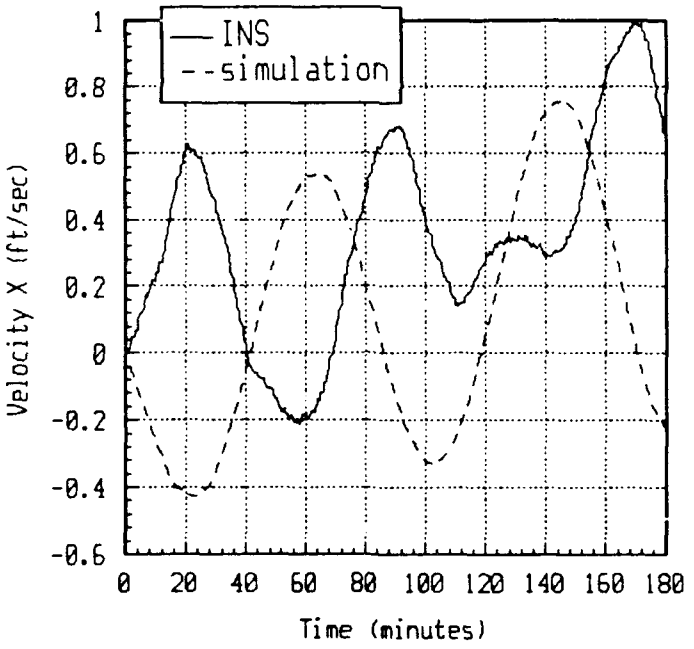


Figure H.5 X Velocity for Simulation and INS-3 Hr Static Navigation

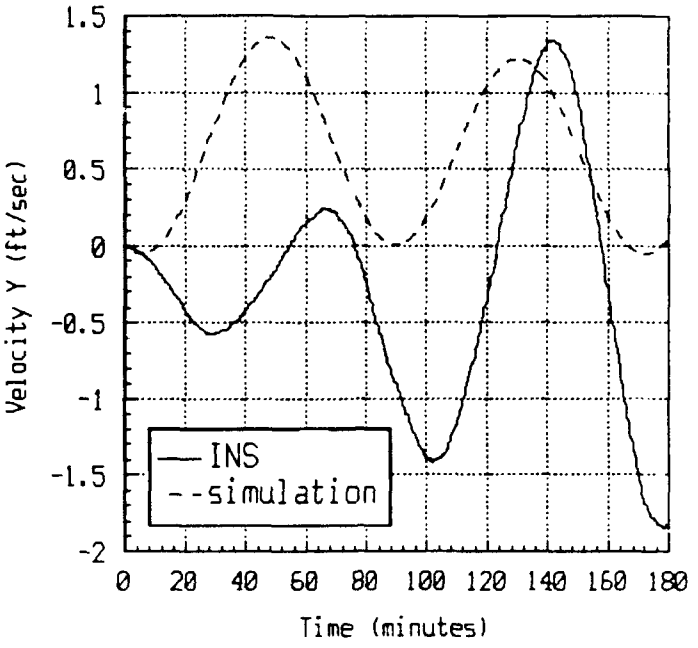


Figure H.6 Y Velocity for Simulation and INS-3 Hr Static Navigation

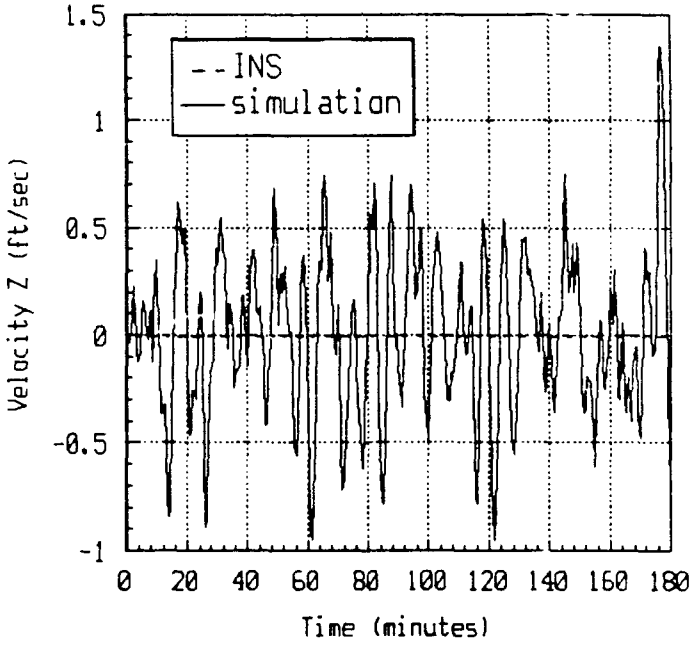


Figure H.7 Z Velocity for Simulation and INS-3 Static Navigation

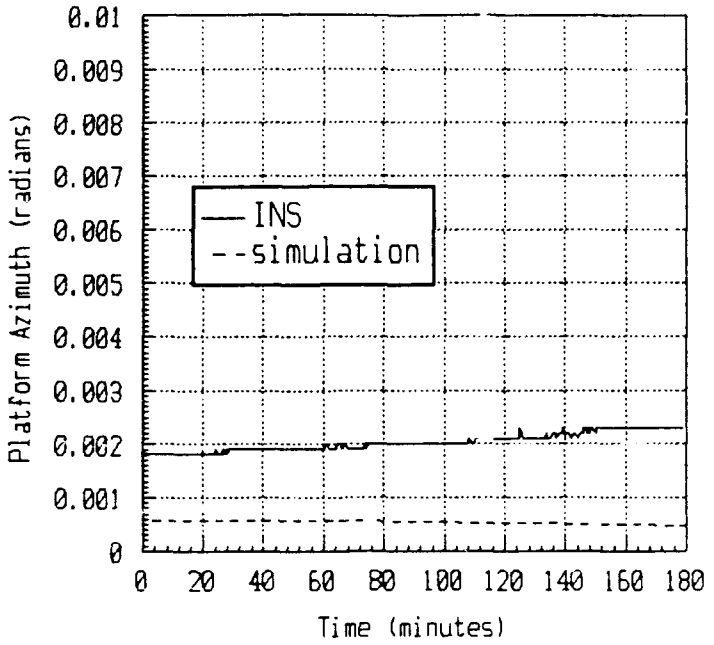


Figure H.8 Platform Azimuth for Simulation and INS-3 Hr Static Navigation

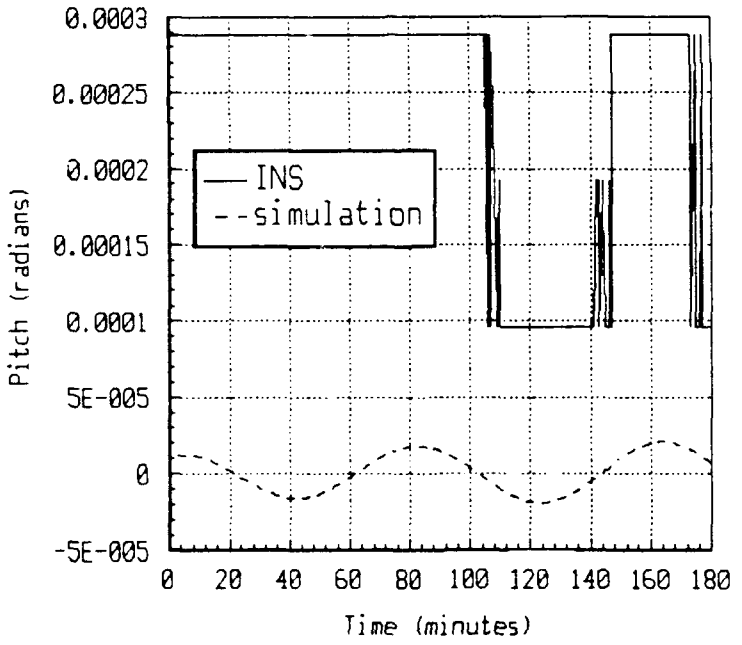


Figure H.9 Pitch for Simulation and INS-3 Hr Static Navigation

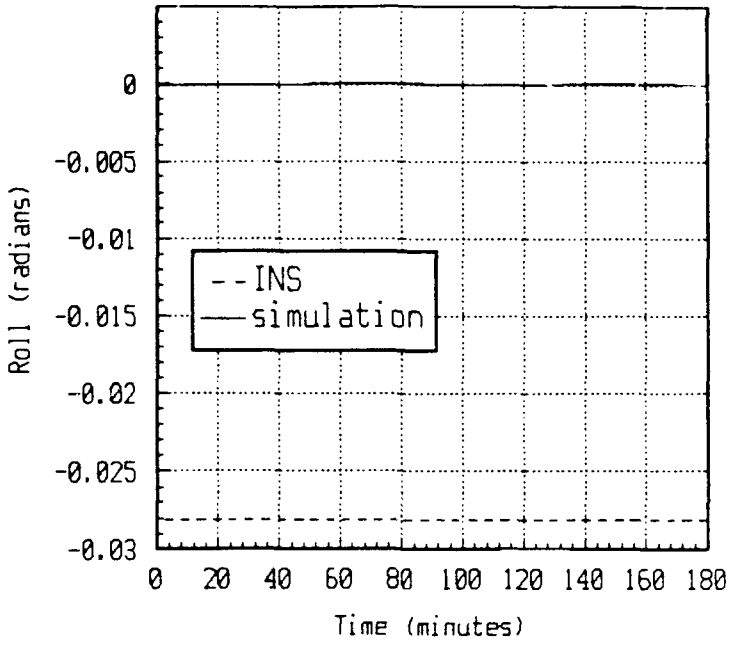


Figure H.10 Roll for Simulation and INS-3 Hr Static Navigation

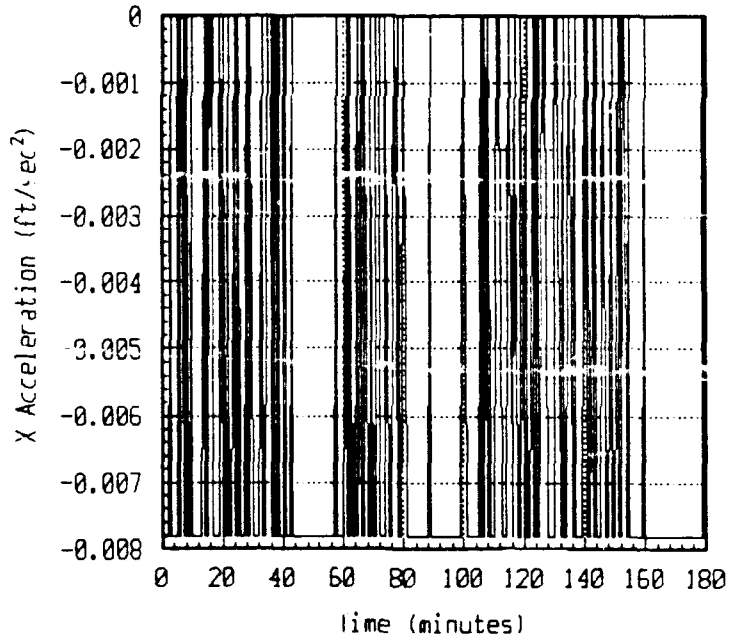


Figure H.11 X Acceleration for INS-3 Hr Static Navigation

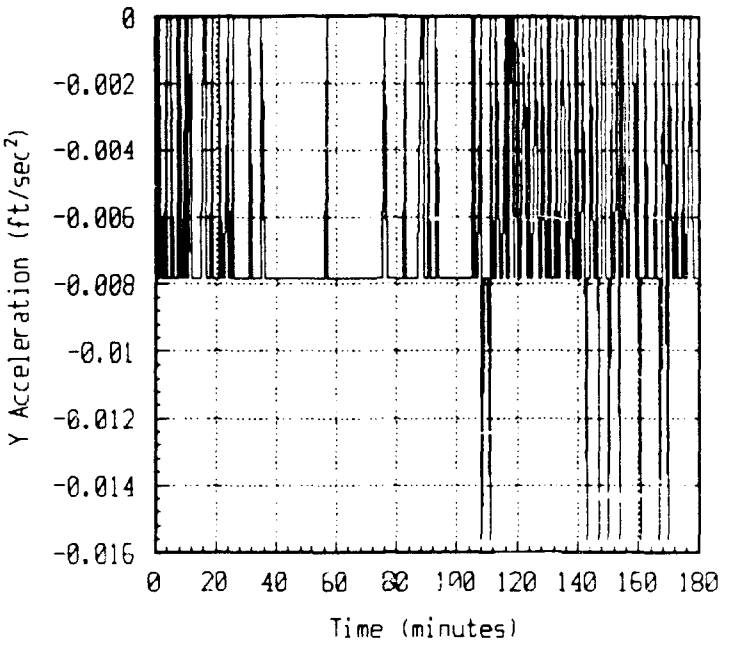


Figure H.12 Y Acceleration for INS-3 Hr Static Navigation

Appendix I: Simulation Validation Results for Straight-and-Level Trajectory

This appendix provides a summary of the plotted results of the straight and level simulated results. Each simulated navigation parameter is plotted against the true trajectory. Where only one curve may appear on the plot, there are two and the error is so insignificant at the scale resolution of the plot as to appear to be zero. Appendix J summarizes the errors between the simulation and the true trajectory.

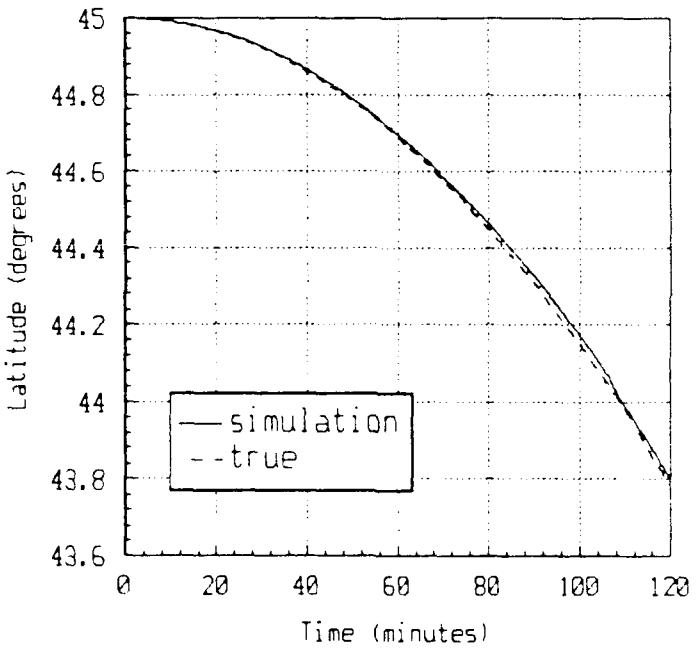


Figure I.1 Latitude of the Simulated and True Straight-and-Level Trajectories

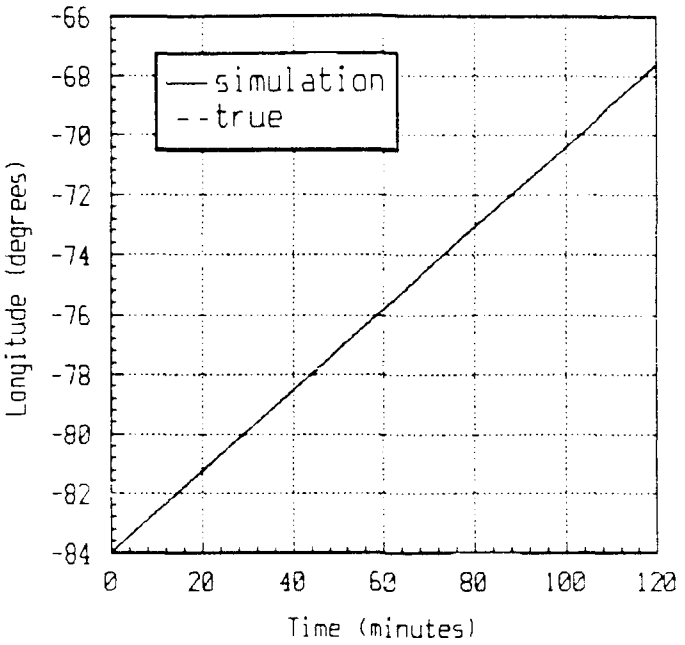


Figure I.2 Longitude of the Simulated and True Straight-and-Level Trajectories

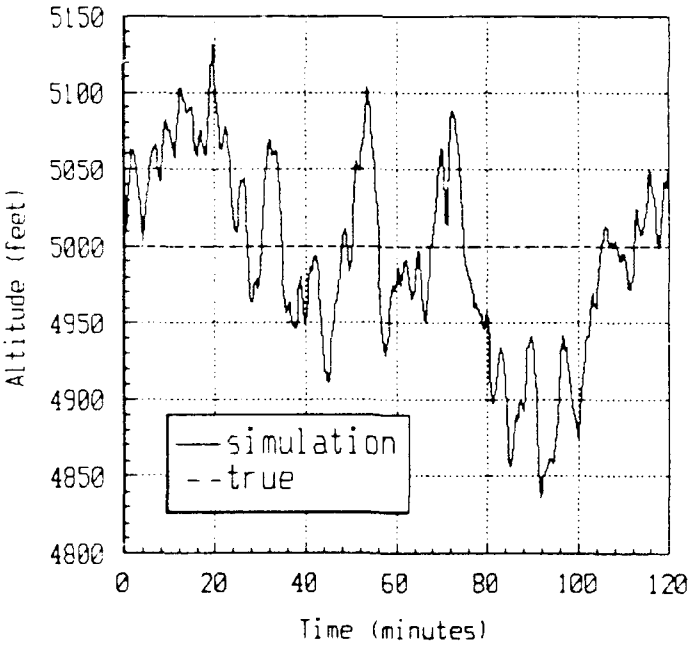


Figure I.3 Altitude of the Simulated and True Straight-and-Level Trajectories

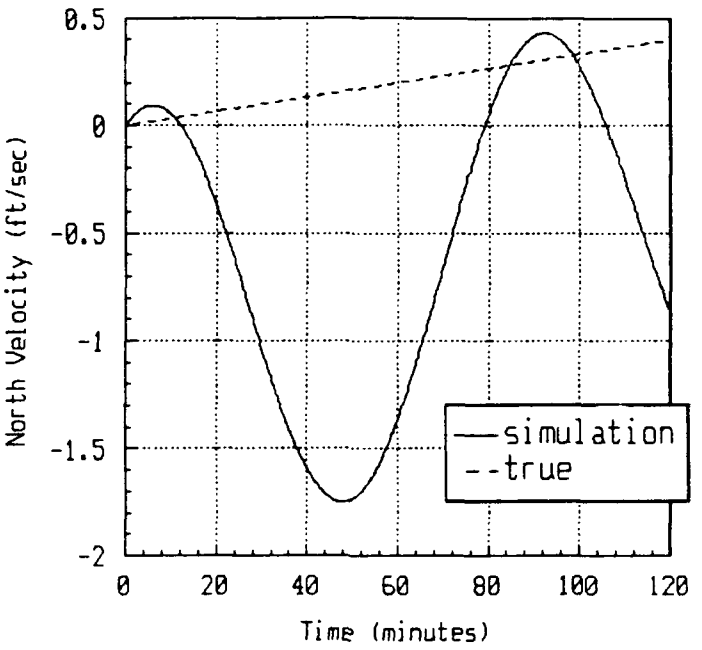


Figure I.4 North Velocity of the Simulated and True Straight-and-Level Trajectories

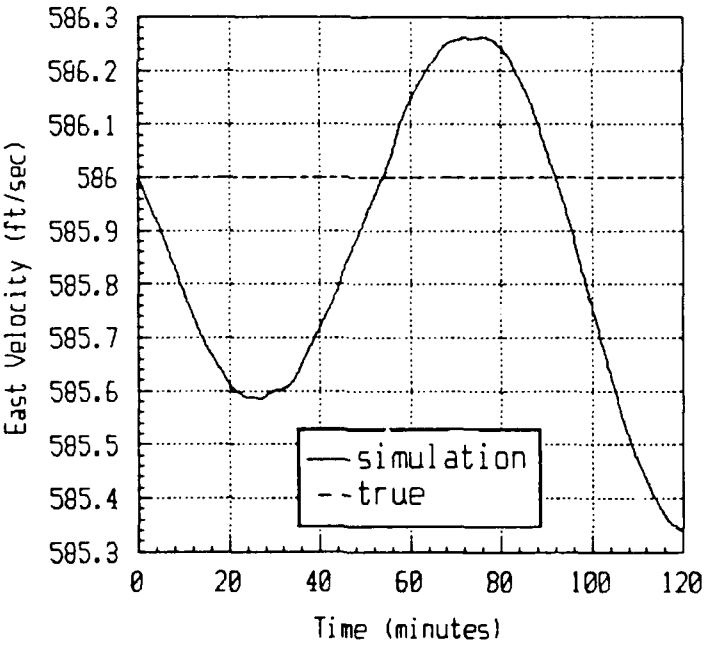


Figure I.5 East Velocity of the Simulated and True Straight-and-Level Trajectories

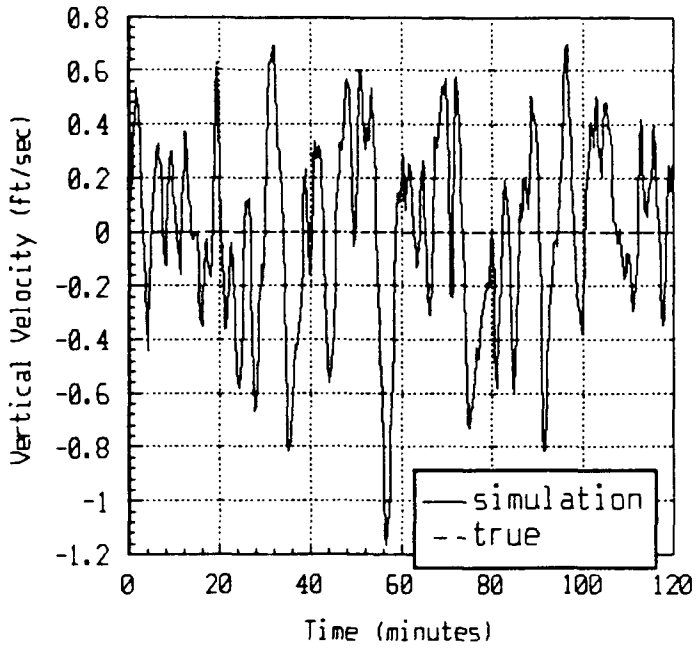


Figure I.6 Vertical Velocity of the Simulated and True Straight-and-Level Trajectories

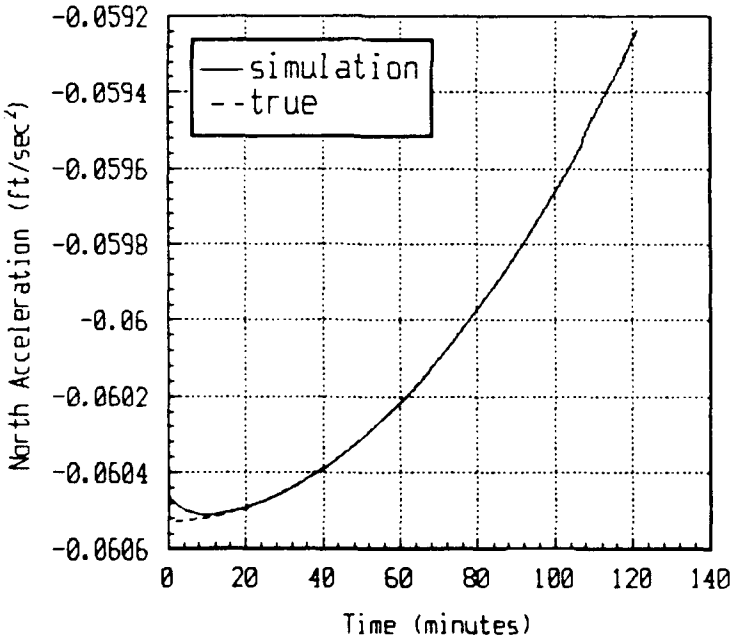


Figure I.7 North Acceleration of the Simulated and True Straight-and-Level Trajectories

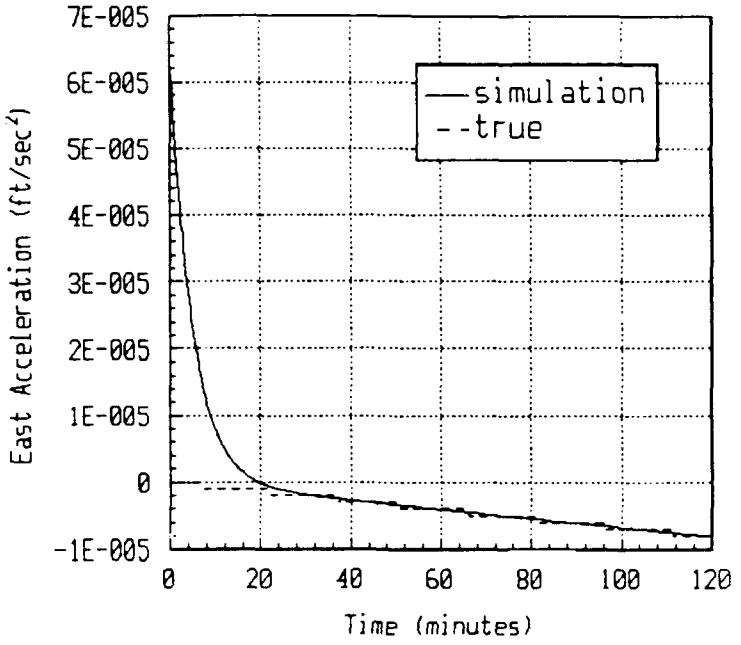


Figure I.8 East Acceleration of the Simulated and True Straight-and-Level Trajectories

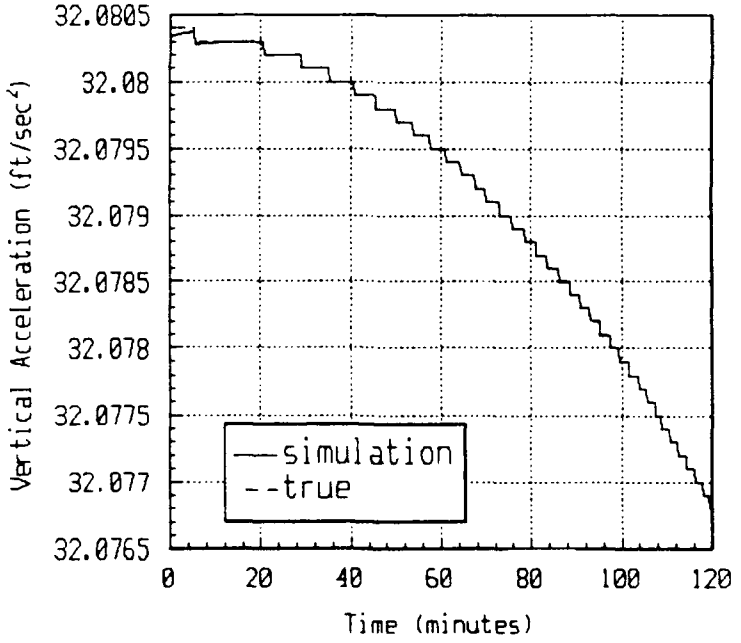


Figure I.9 Vertical Acceleration of the Simulated and True Straight-and-Level Trajectories

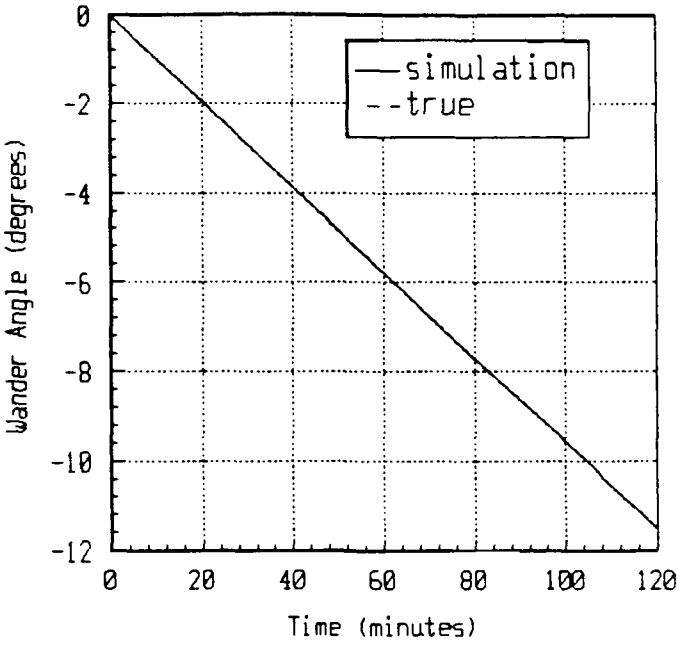


Figure I.10 Wander Angle of the Simulated and True Straight-and-Level Trajectories

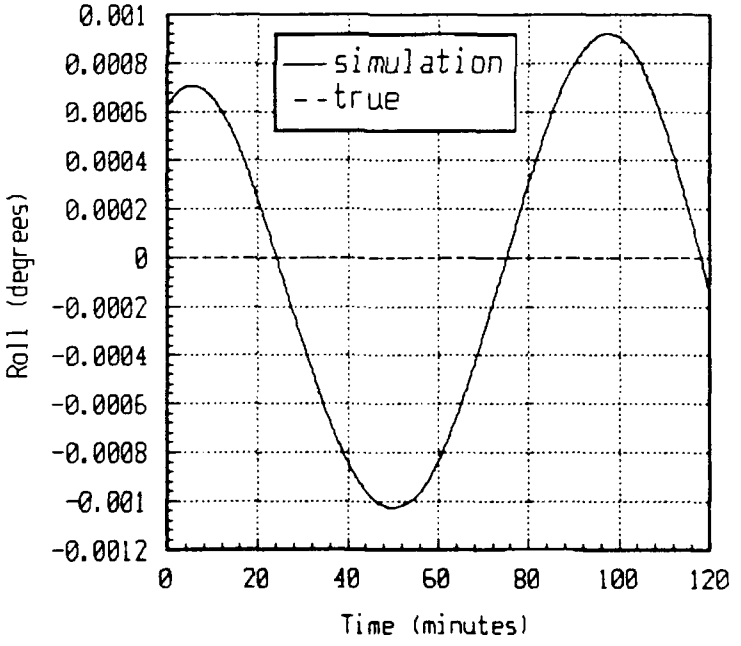


Figure I.11 Roll of the Simulated and True Straight-and-Level Trajectories

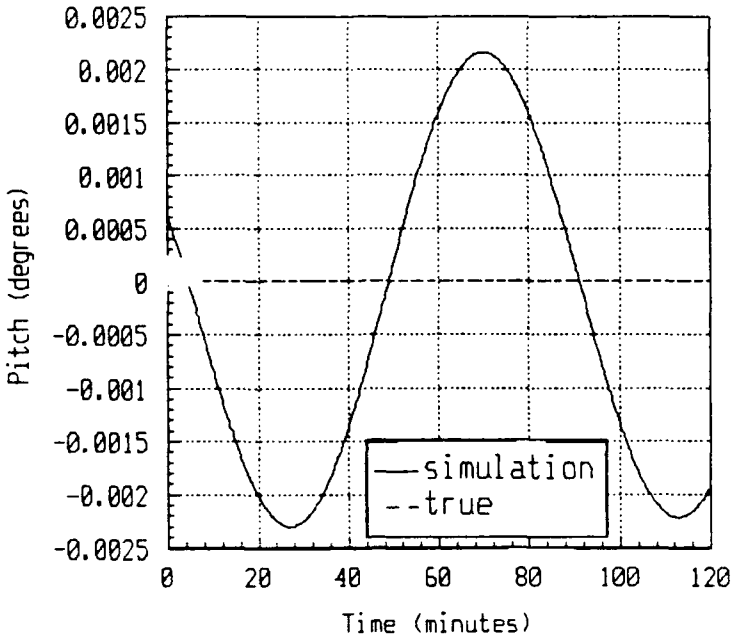


Figure I.12 Pitch of the Simulated and True Straight-and-Level Trajectories

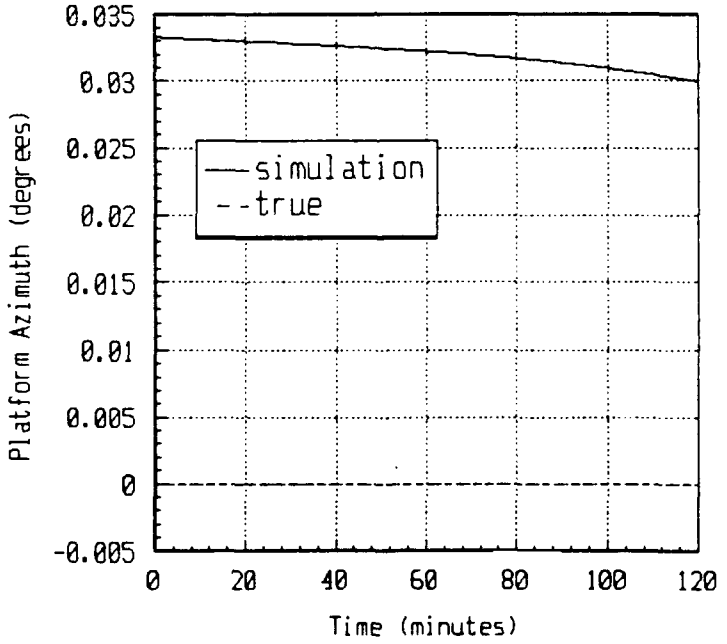


Figure I.13 Platform Azimuth of the Simulated and True Straight-and-Level Trajectories

Appendix J: Error Behavior of Simulated Straight-and-Level Trajectory

This appendix provides a summary of the error behavior of the simulation relative to a true straight-and-level trajectory. The results of the simulation are differenced with the true trajectory and the results are plotted for analysis. Appendix I provides the plots of data that are used to develop these error results.

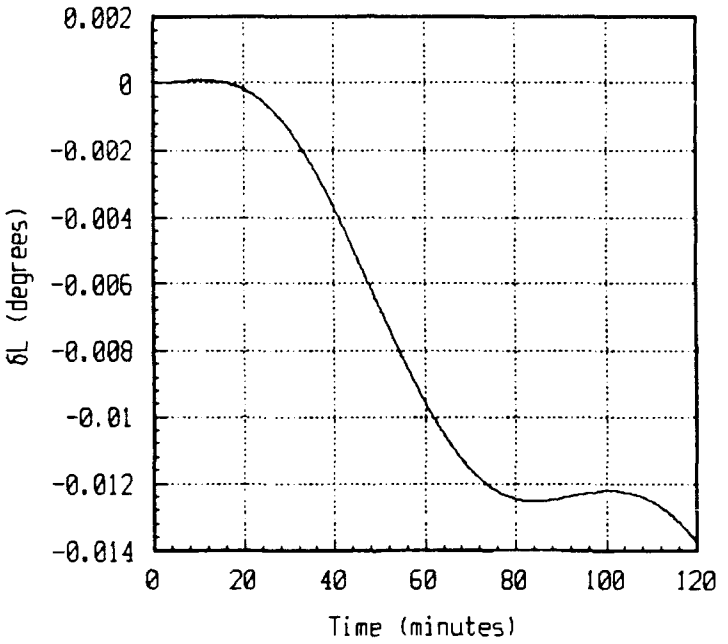


Figure J.1 Latitude Error in the Simulated and True Straight-and-Level Trajectories

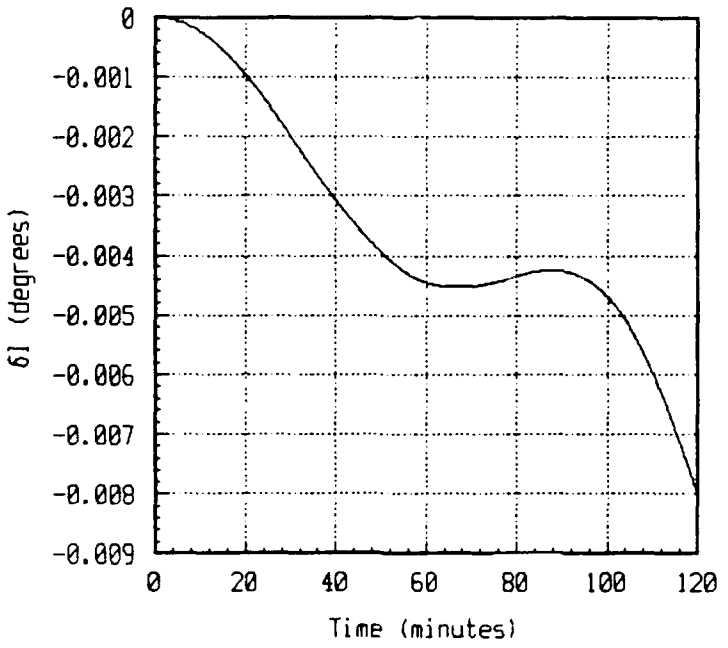


Figure J.2 Longitude Error in the Simulated and True Straight-and-Level Trajectories

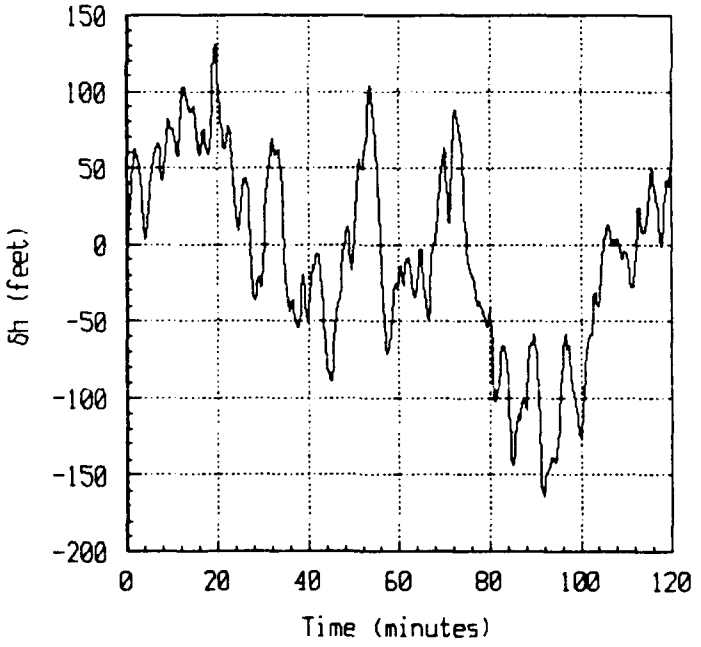


Figure J.3 Altitude Error in the Simulated and True Straight-and-Level Trajectories

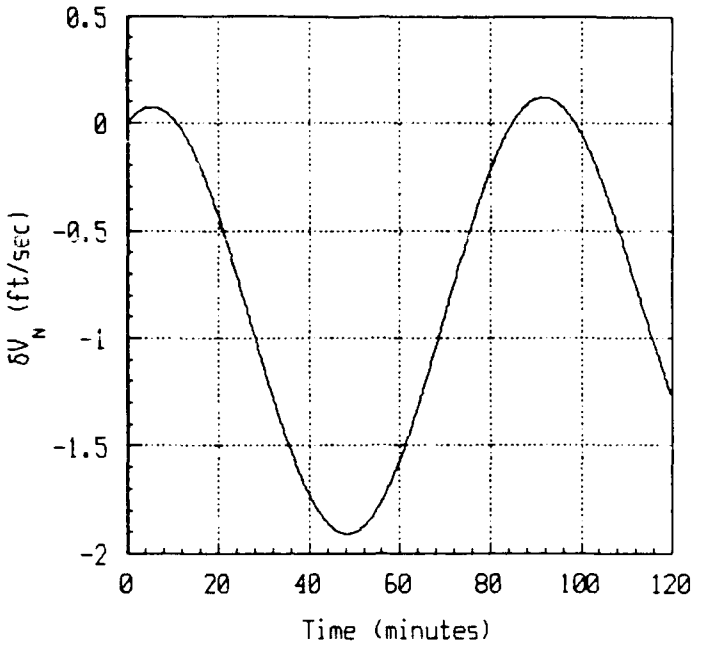


Figure J.4 North Velocity Error in the Simulated and True Straight-and-Level Trajectories

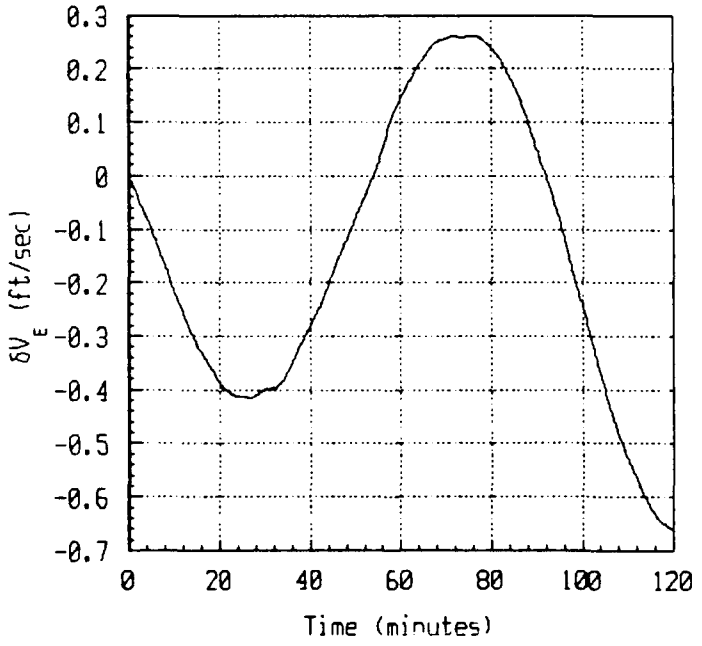


Figure J.5 East Velocity Error in the Simulated and True Straight-and-Level Trajectories

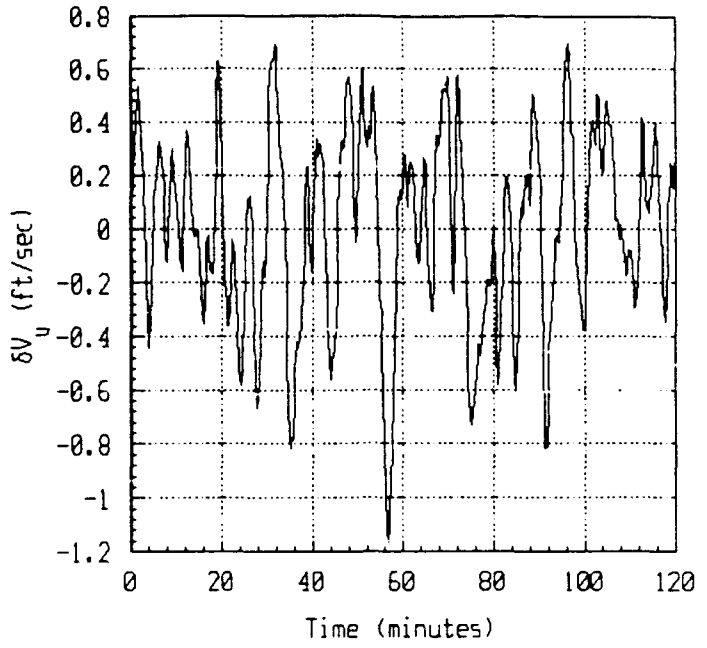


Figure J.6 Vertical Velocity Error in the Simulated and True Straight-and-Level Trajectories

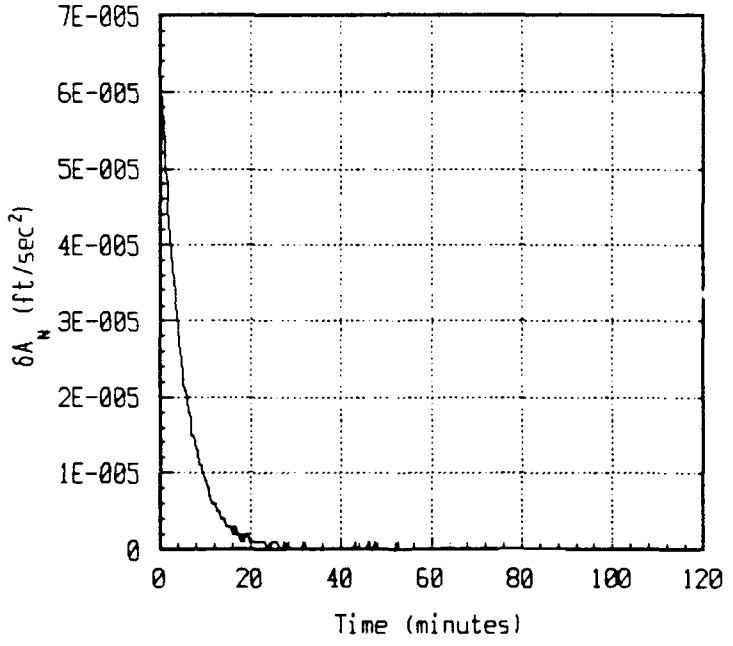


Figure J.7 North Acceleration Error in the Simulated and True Straight-and-Level Trajectories

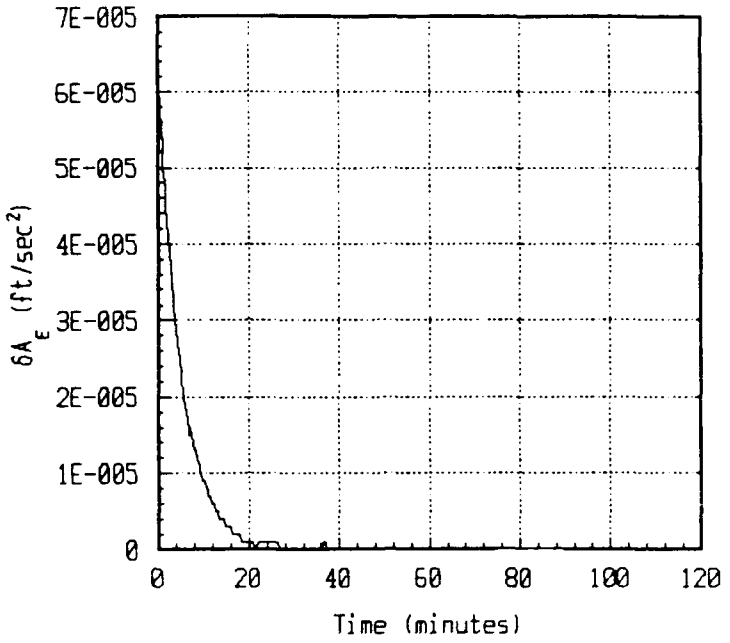


Figure J.8 East Acceleration Error in the Simulated and True Straight-and-Level Trajectories

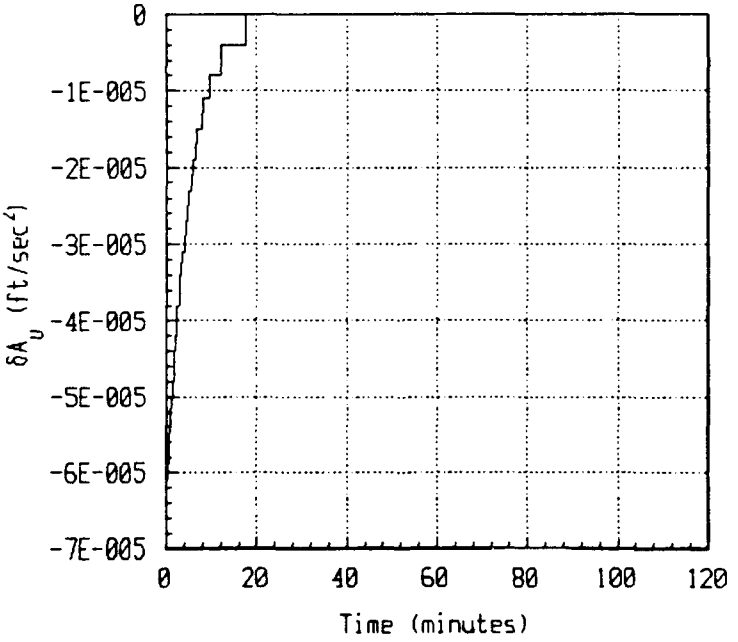


Figure J.9 Vertical Acceleration Error in the Simulated and True Straight-and-Level Trajectories

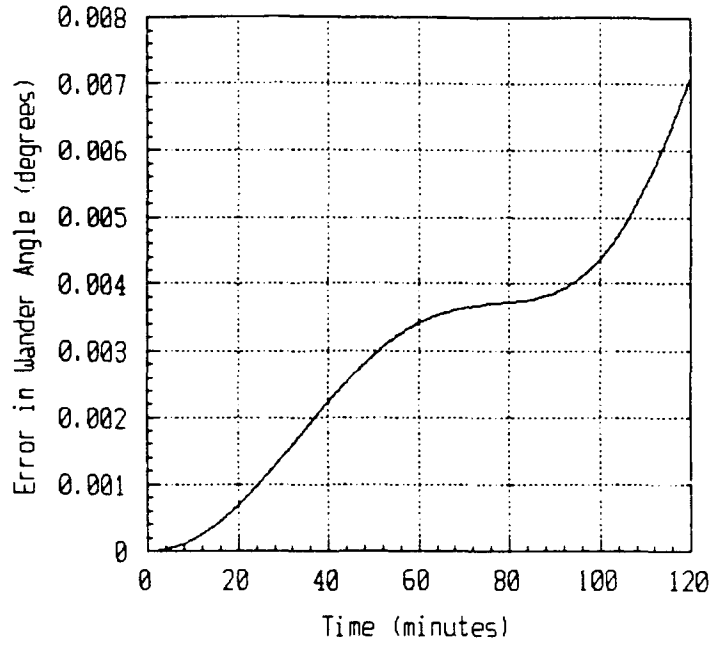


Figure J.10 Wander Angle Error in the Simulated and True Straight-and-Level Trajectories

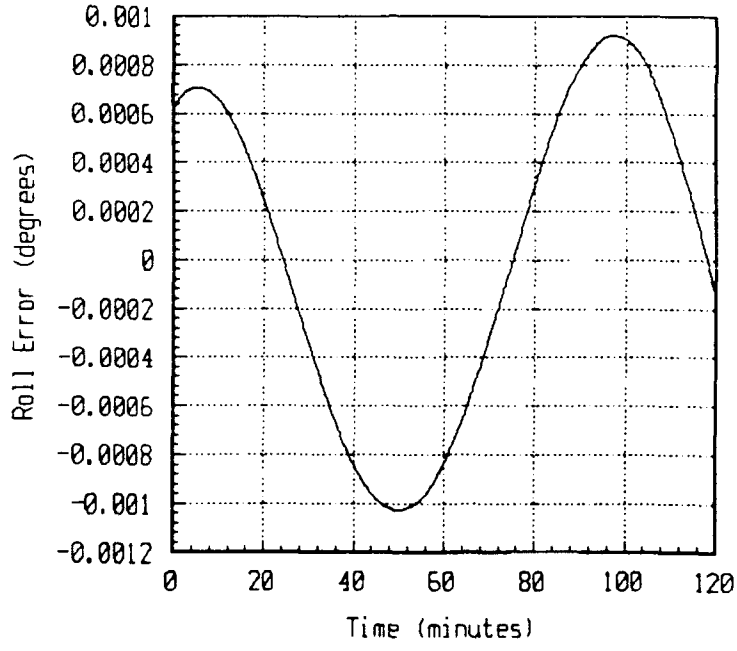


Figure J.11 Roll Error in the Simulated and True Straight-and-Level Trajectories

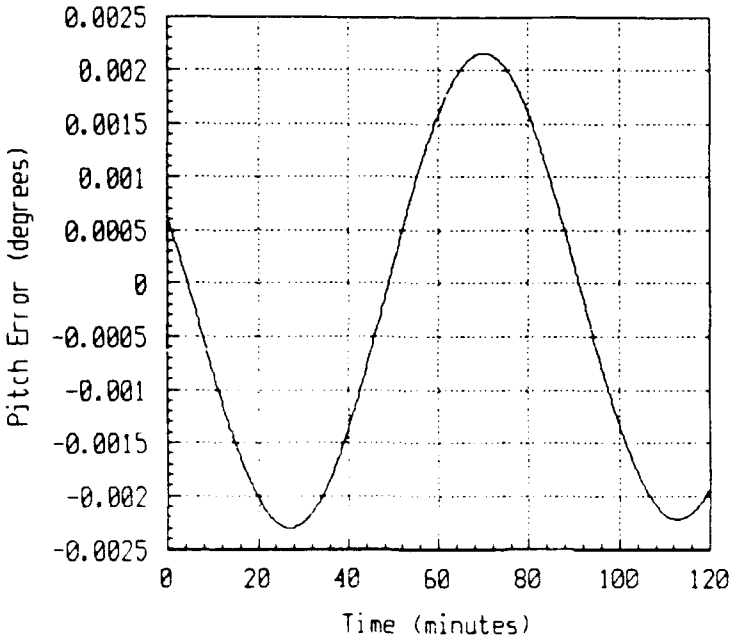


Figure J.12 Pitch Error in the Simulated and True Straight-and-Level Trajectories

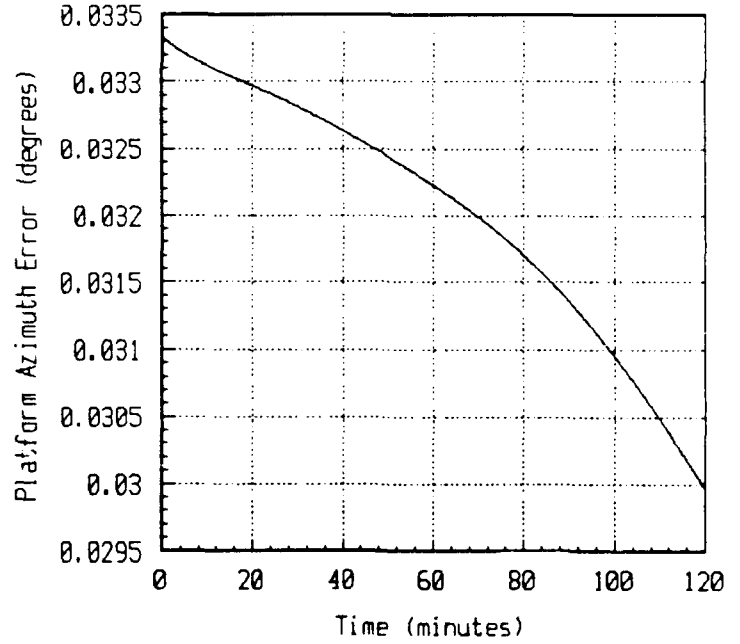


Figure J.13 Platform Azimuth Error in the Simulated and True Straight-and-Level Trajectories

Bibliography

1. Academia, Janet. Programmer, Intermetrics Inc., Source Code for INS Simulation Program. Version 1.9, Huntington Beach CA., 2 June 1989.

2. Air Force Avionics Laboratory. PROFGEN - A Computer Program for Generating Flight Profiles, Technical Report AFAL-TR-76247, November 1976. AFAL/AAAN-2, Wright-Patterson AFB, OH.

3. ASD, AFSC, USAF. Specification for USAF F-15 Inertial Navigation Set. Technical Report FNU 85-1. Revision A Change Notice 3. Wright-Patterson AFB, OH:HQ AFSC, April 1988.

4. ASD, AFSC, USAF. Specification for USAF Standard Form, Fit and Function Medium Accuracy Inertial Navigation Unit. Technical Report SNU 84-1. Revision 3 AMD 2. Wright-Patterson AFB, OH:HQ AFSC, April 1988.

5. Ballard Technology. PC1553-2 User's Manual. Seattle:Ballard Technology, Rev A, Sep 15, 1988.

6. Barclay, Kenneth. Engineer, Personal Interviews, Intermetrics Inc., Huntington Beach CA., July 90 thru Sep 90.

7. Breaux, Harold J. An Efficiency Study of Several Techniques for the Numerical Integration of the Equations of Motion for Missiles and Shell. Report No 1358. Aberdeen Proving Ground Maryland: Ballistics Research Laboratory, February 1967 (AD 812362)

8. Britting, Kenneth R. Inertial Navigation System Analysis. New York:Wiley-Interscience, 1971.

9. Carlson, Neal A. and others Program Design Description for a Multimode Simulation for Optimal Filter Evaluation (MSOFE). Technical Report AFWAL-TR-88-1137, February 1989. AFWAL/AAAN-2, Wright-Patterson AFB, OH.

10. Carlson, Neal A. and others User's Manual for a Multimode Simulation for Optimal Filter Evaluation (MSOFE). Technical Report AFWAL-TR-88-1138, July 1989. AFWAL/AAAN-2, Wright-Patterson AFB, OH.

11. C Utility Library User Guide. Version 3, Maplewood New Jersey: Essential Software Inc.

12. Defense Mapping Agency. Department of Defense World Geodetic System 1984. Defense Mapping Agency Technical Report, No DMA TR 8350.2. Washington, D.C., 30 September 1987.

13. Digital Technology Inc. User's Manual-SCE4 (MS-DOS) Application Software for the DTI-1120. Dayton, OH:Digital Technology INC, 1990.

14. Hirning, James L. Development of a Kalman Filter to Optimally Integrate a Global Positioning System Receiver and an LN 94 Inertial Navigation System. Thesis, September 1990.

15. Inertial Navigation Covariance Analysis (INERCA) User Manual. GRC-1521-03-1, Ft Walton Beach Florida: General Research Corporation, February 1989.

16. Kayton, Myron and Walter Fried. Avionics Navigation Systems. New York:Wiley and Sons, Inc., 1969.

17. Litton Guidance and Control Systems. Performance Accuracy (Truth Model/Error Budget) Analysis for the LN-93 Inertial Navigation Unit. DID No. DI-S-21433 B/T: CDRL No. 1002. Woodland Hills, CA., January 1985.

18. Maybeck, Peter S. Stochastic Models, Estimation, and Control Volume 1. New York:Academic Press, 1979.

19. Maybeck, Peter S. Professor, Department of Electrical and Computer Engineering, Personal Interviews, Air Force Institute of Technology, Wright-Patterson AFB, OH., July 90 thru May 91.

20. Microsoft Corporation. Microsoft C Professional Development System. Version 6.0, Redmond WA:Microsoft Corporation, 1990.

21. Microsoft Corporation. Microsoft Macro Assembler. Version 5.1, Redmond WA:Microsoft Corporation, 1987.

22. Mil-Std-1553 Designer's Guide. Second Edition, Second Printing New York:ILC Data Device Corporation, 1988.

23. Paschall, R. Assistant Professor, Department of Electrical and Computer Engineering, Personal Interviews, Air Force Institute of Technology, Wright-Patterson AFB, OH., July 90 thru May 91.

24. Press, William H. and others. Numerical Recipes in C-The Art of Scientific Computing. New York:Cambridge University Press, 1990.

25. Reid, Gary J. Linear System Fundamentals-Continuous and Discrete Classic and Modern New York:McGraw-Hill, 1983.

26. Stacey, Richard D. A Navigation System (NRS) Using Global Positioning System (GPS) and Transponder Aiding. Thesis, March 1991.

27. Stacey, Richard D. Graduate Student, Department of Electrical and Computer Engineering, Personal Interviews, Air Force Institute of Technology, Wright-Patterson AFB, OH., July 90 thru March 91.

28. WATCOM Group Inc. WATCOM C Optimizing Compiler and Tools. Version 2.0, Waterloo Ontario:WATCOM Publications, 1989.